

**VISVESVARAYA TECHNOLOGICAL
UNIVERSITY**
“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT

on

Machine Learning (23CS6PCMAL)

Submitted by

Agneya D A (1BM22CS024)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Sep-2024 to Jan-2025

**B.M.S. College of Engineering,
Bull Temple Road, Bangalore 560019**
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the Lab work entitled “Machine Learning (23CS6PCMAL)” carried out by **Agneya D A (1BM22CS024)**, who is bonafide student of **B.M.S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Lab faculty Incharge Name: M Lakshmi Neelima Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE
--	--

Index

Sl. No.	Date	Experiment Title	Page No.
1	4-3-2025	Write a python program to import and export data using Pandas library functions	4
2	11-3-2025	Demonstrate various data pre-processing techniques for a given dataset	10
3	18-3-2025	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.	24
4	1-4-2025	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset	38
5	8-4-2025	Build Logistic Regression Model for a given dataset	47
6	15-4-2025	Build KNN Classification model for a given dataset.	65
7	15-4-2025	Build Support vector machine model for a given dataset	77
8	22-4-2025	Implement Random forest ensemble method on a given dataset.	89
9	22-4-2025	Implement Boosting ensemble method on a given dataset.	97
10	29-4-2025	Build k-Means algorithm to cluster a set of data stored in a .CSV file.	104
11	29-4-2025	Implement Dimensionality reduction using Principal Component Analysis (PCA) method.	113

Github Link:

https://github.com/Agneya-1BM22CS024/ML_Lab

Program 1

Write a python program to import and export data using Pandas library functions
Algorithm:

LAB - 2

DATE: 4/3

Insert To Do - 1

```
import pandas as pd
i df = pd.read_csv('housing.csv') # Load data from
ii print(df.head(2)) # Display data of all values
iii print(df.describe()) # Display statistical information of all nu
iv print(df['ocean-proximity'].value_counts()) # Count o
v print(df['rooms'].sum())
```

Output:

ii longitude latitude housing_median-age total_rooms

	longitude	latitude	housing_median-age	total_rooms
0	-122.23	37.88	41.0	880.0
1	-122.22	37.86	21.0	2099.0

iii longitude latitude housing_median-age total_rooms

	longitude	latitude	housing_median-age	total_rooms
mean	-119.569	35.632	28.639	2635

Population households median-income median-housing

	Population	Households	Median-Income	Median-Housing
mean	1425.476	499.539	43.875	206

iv Ocean_proximity

	Ocean_Proximity	Count
<1H OCEAN	9136	
INLAND	6551	
NEAR OCEAN	2658	
NEAR BAY	2290	
ISLAND	5	

Name: count, dtype: int64

NL
4/3/2021

households	0
median_income	0
median_house_value	0
ocean_proximity	0
dtype: int64	

v # To display which attributes in a dataset have missing values

missing = df.isna().sum()

missing_2 = df[missing[missing > 0]]

print(missing_2)

Output:

total_bedrooms 207

dtype: int64

To do - 2

Questions:

1. Which columns in the dataset had missing values? You handle them.
2. Which categorical columns did you identify in? How did you encode them?
3. What is the difference between Min-Max Scaling and Standardization? When would you use one over the other?

Diabetes

1. There are no missing values.

DATE: 9

Standardization

Transforms data to have zero mean & unit std.

$$x' = \frac{x - \bar{x}}{\sigma}$$

f_t is used when a dataset follows a Gaussian distribution in my algorithms which assume normality.

Adult Income

1. missing values are represented by "?" which is np.nan.

For numerical columns

$$df.fillna(\{"column": df["column"].mean()\})$$

For categorical data

$$df.fillna(\{"column": df["column"].mode()\})$$

2. Work-class, education, marital-status, occupation, relative-native-country, income are the categorical columns. We use label encoding to convert categorical data to numerical format.

for col in df.select_dtypes(include=['object']).columns:

 le = LabelEncoder()

~~df[col] = le.fit_transform(df[col])~~
~~label_encoder[col] = le~~

Code:

```
import pandas as pd  
  
df= pd.read_csv('housing.csv')  
  
df.columns
```

Output:

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',  
       'total_bedrooms', 'population', 'households', 'median_income',  
       'median_house_value', 'ocean_proximity'],  
      dtype='object')  
df.describe()
```

Output:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
co	20640.00000	20640.00000	20640.00000	20640.00000	20433.00000	20640.00000	20640.00000	20640.00000	20640.00000
un	0	0	000	0	00000	0	0	00000	000
t									
m	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
ea									
n									
st	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
d									
mi	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.00000
n									
25	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.00000
%									
50	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.00000
%									
75	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.00000
%									

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
max	-114.310000	41.950000	52.000000	39320.00000	6445.00000	35682.00000	6082.00000	15.000100	500001.00000

```
l=df.ocean_proximity.unique()
```

```
print(l)
```

```
print(len(l))
```

```
['NEAR BAY' '<1H OCEAN' 'INLAND' 'NEAR OCEAN' 'ISLAND']
```

```
5
```

```
df.isna().sum()
```

```
print(df.columns[df.isnull().any()])
```

```
Index(['total_bedrooms'], dtype='object')
```

```
df=pd.read_csv('/content/Dataset of Diabetes .csv')
```

```
print("Sample Data")
```

```
print(df.head())
```

```
print(df.isna().sum())
```

```
print(df.columns[df.isnull().any()])
```

```
categorical_cols = df.select_dtypes(exclude=['number']).columns
```

```
categorical_cols
```

Sample Data

```
ID No_Pation Gender AGE Urea Cr HbA1c Chol TG HDL LDL VLDL \
0 502 17975 F 50 4.7 46 4.9 4.2 0.9 2.4 1.4 0.5
1 735 34221 M 26 4.5 62 4.9 3.7 1.4 1.1 2.1 0.6
2 420 47975 F 50 4.7 46 4.9 4.2 0.9 2.4 1.4 0.5
3 680 87656 F 50 4.7 46 4.9 4.2 0.9 2.4 1.4 0.5
4 504 34223 M 33 7.1 46 4.9 4.9 1.0 0.8 2.0 0.4
```

BMI CLASS

```
0 24.0 N
```

```
1 23.0 N
```

```
2 24.0 N
```

```
3 24.0 N
```

```
4 21.0 N
```

```
ID 0
```

```
No_Pation 0
```

```
Gender 0
```

```
AGE 0
```

```
Urea 0
```

```

Cr      0
HbA1c   0
Chol    0
TG      0
HDL     0
LDL     0
VLDL    0
BMI     0
CLASS   0
dtype: int64
Index([], dtype='object')
Out[42]:
Index(['Gender', 'CLASS'], dtype='object')
In [44]:

```

```

df=pd.read_csv('/content/adult.csv')
print("Sample Data")
print(df.head())
print(df.isna().sum())
print(df.columns[df.isnull().any()])
categorical_col = df.select_dtypes(exclude=['number']).columns
categorical_col

```

Sample Data

	age	workclass	fnlwgt	education	educational-num	marital-status
0	25	Private	226802	11th	7	Never-married
1	38	Private	89814	HS-grad	9	Married-civ-spouse
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse
3	44	Private	160323	Some-college	10	Married-civ-spouse
4	18	?	103497	Some-college	10	Never-married

	occupation	relationship	race	gender	capital-gain	capital-loss
0	Machine-op-inspct	Own-child	Black	Male	0	0
1	Farming-fishing	Husband	White	Male	0	0
2	Protective-serv	Husband	White	Male	0	0
3	Machine-op-inspct	Husband	Black	Male	7688	0
4	?	Own-child	White	Female	0	0

	hours-per-week	native-country	income
0	40	United-States	<=50K
1	50	United-States	<=50K
2	40	United-States	>50K
3	40	United-States	>50K
4	30	United-States	<=50K

age	0
workclass	0
fnlwgt	0

```
education      0
educational-num 0
marital-status  0
occupation     0
relationship    0
race           0
gender          0
capital-gain    0
capital-loss    0
hours-per-week  0
native-country   0
income          0
dtype: int64
Index([], dtype='object')
Output:
Index(['workclass', 'education', 'marital-status', 'occupation',
       'relationship', 'race', 'gender', 'native-country', 'income'],
      dtype='object')
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Algorithm:

Lab - 1

i. Initializing values directly into dataframe

= import pandas as pd

data = {

'Name': ['Alice', 'Bob', 'Charlie', 'David'],

'USN': ['001', '002', '003', '004']

'Marks': [25, 30, 35, 40]

}

df = pd.DataFrame(data)

print(df)

Out:

	USN	Name	Marks
0	001	Alice	25
1	002	Bob	30
2	003	Charlie	35
3	004	David	40

ii. From sklearn.datasets

= import pandas as pd

from sklearn.datasets import load_diabetes

import pandas as pd

iris = load_diabetes

df = pd.DataFrame(data=iris.data, columns=iris

df['target'] = iris.target

print(df.head)

Out:

	age	sex	bmi	bp	s1	s2	s3
0	0.038	0.050	0.061	0.071	-0.044	-0.034	-0.04

DATE: 9/3/

iii From CSV dataset

= File-path = "/content/campsite-sales-data.csv"
df = pd.read_csv(file-path)
print(df.head(2))

Out:

	Product	Quality	Price	Sales	Region
0	Laptop	5	1000	5000	North

iv Downloading data from existing dataset repositories

= df = pd.read_csv("https://raw.githubusercontent.com/DiabetesDataset/Diabetes/master/diabetes.csv")
df.head(1)

Out:

	IP	No_Pregn	Gender	Age	Urea	G	HbA1C	Chol	TG	HDL	LDL	S
0	502	12705	F	50	4.7	46	49	4.2	0.9	3.4	1	1

Bank Data Analysis

import pandas as pd

import yfinance as yf

import matplotlib.pyplot as plt

tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]

data = yf.download(tickers, start="2024-01-01", end="2024-01-01")
= "ticker")

print(data.head(1))

Out:

Ticker HDFCBANK.NS

Price	open	high	low	close	volume
-------	------	------	-----	-------	--------

DATE: 4/3/35

2024-01-01 1906.969 1916.899 1891.627 1907.059 1426.9

Ticker ICICI BANK.NS

Date	Open	High	Low	Close	Volume
2024-01-01	1986.929	1996.273	1982.541	1990.869	768375

HDFC

```
hdfc_data = data[['HDFC.NS']]
```

```
plt.figure(figsize=(12,6))
plt.subplot(2,1,1)
```

```
hdfc_data['close'].plot(title='HDFC closing price')
```

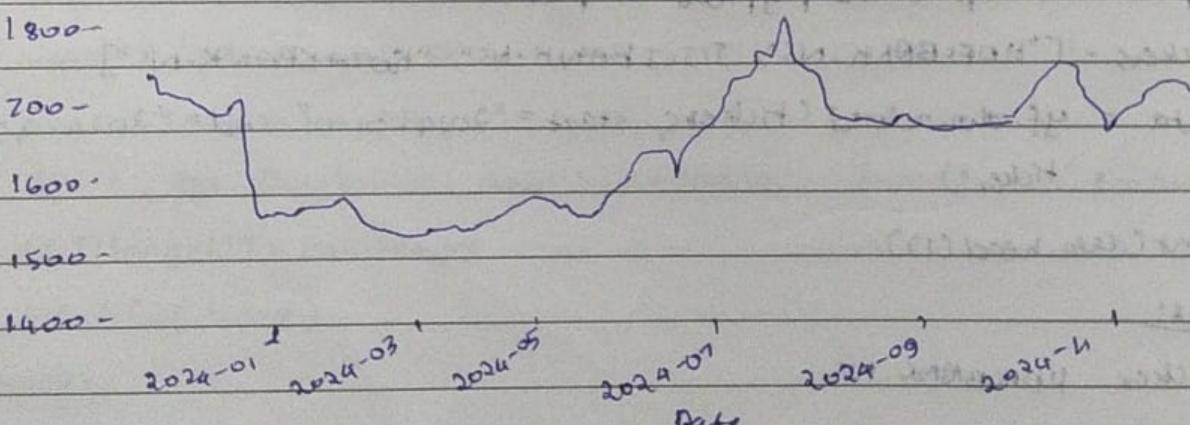
```
plt.subplot(2,1,2)
```

```
hdfc_data['Daily Return'].plot(title='HDFC Daily Return')
```

```
plt.tight_layout()
```

```
plt.show()
```

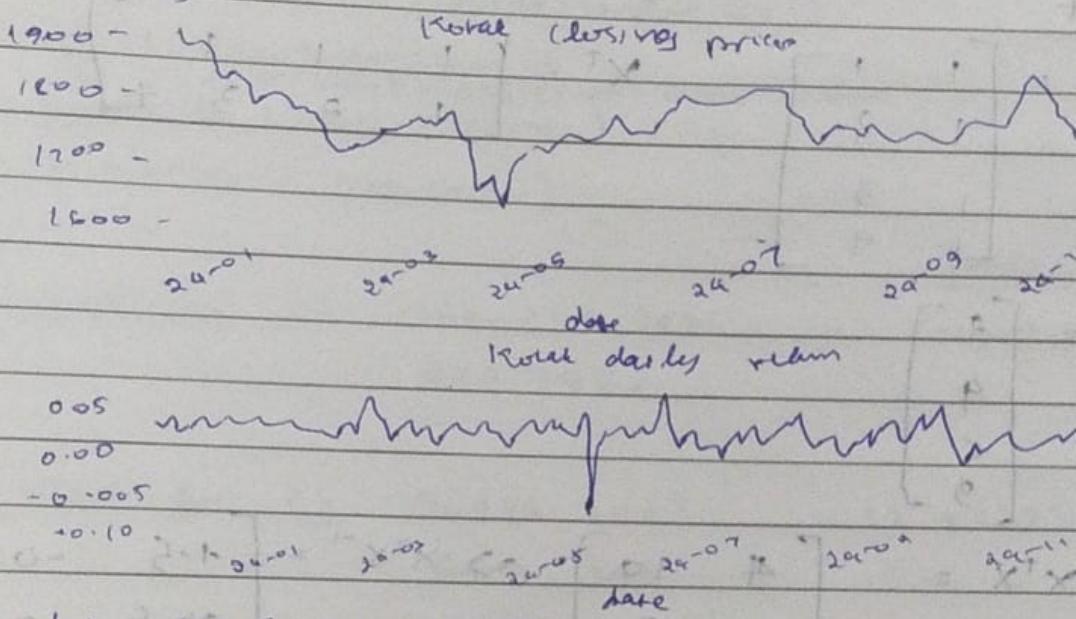
HDFC Closing Price



HDFC Daily Return

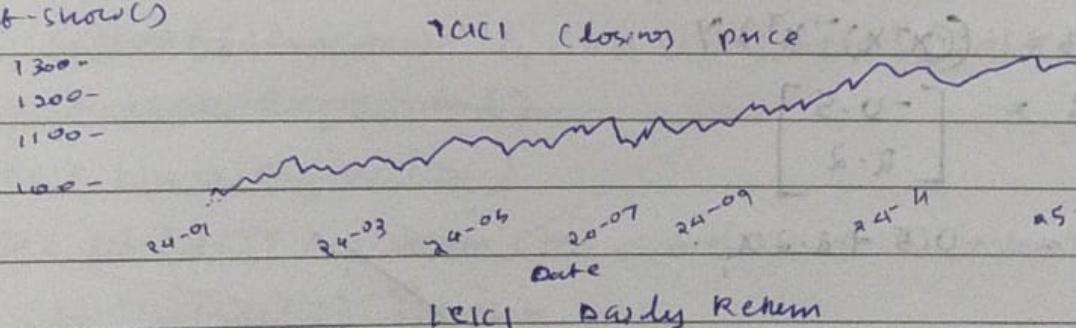
DATE: 7/

```
kotak_data = data['KOTAKBANK.NS']
kotak_data['Daily Return'] = kotak_data['(close)'].pct_change()
plt.figure(figsize=(12,6))
kotak_data['(close)'].plot(title='Kotak Closing Price')
plt.subplot(2,1,2)
kotak_data['Daily Return'].plot(title="Kotak Daily Return")
plt.show()
```



```
icici_data = data['ICICIBANK.NS']
icici_data['Daily Return'] = icici_data['close'].pct_change()
plt.subplot(2,1,1)
```

```
icici_data['(close)'].plot(title='ICICI Closing Price')
plt.subplot(2,1,2)
icici_data['Daily Return'].plot(title='ICICI Daily Return')
plt.tight_layout()
plt.show()
```



Code:

```
import pandas as pd
data ={
    'Name':['Alice','Bob','Charlie','David'],
    'Age':[25,30,35,40],
    'City':['New York','Los Angeles','Chicago','Houston']
}
data = pd.DataFrame(data)
print(data)
```

```
Name  Age      City
0   Alice  25  New York
1     Bob  30  Los Angeles
2  Charlie  35    Chicago
3   David  40    Houston
```

In []:

```
from sklearn.datasets import load_iris
iris= load_iris()
df=pd.DataFrame(iris.data,columns=iris.feature_names)
df['target']=iris.target
df.head()
print(df)
```

```
sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm) \
0            5.1          3.5           1.4           0.2
1            4.9          3.0           1.4           0.2
2            4.7          3.2           1.3           0.2
3            4.6          3.1           1.5           0.2
4            5.0          3.6           1.4           0.2
..           ...
145           6.7          3.0           5.2           2.3
146           6.3          2.5           5.0           1.9
147           6.5          3.0           5.2           2.0
148           6.2          3.4           5.4           2.3
149           5.9          3.0           5.1           1.8
```

```
target
0     0
1     0
2     0
3     0
4     0
..   ...
145    2
146    2
```

```
147    2
148    2
149    2
```

[150 rows x 5 columns]

In []:

```
file_path='data.csv'
df=pd.read_csv(file_path)
print("Sample Data")
print(df.head())
```

Sample Data

```
ID  Name  Age   City
0  1   Alice  25  New York
1  2   Bob   30  Los Angeles
2  3   Charlie  35  Chicago
3  4   David  40  Houston
4  5   Eva   28  Phoenix
```

In [20]:

```
df=pd.read_csv('mobiles.csv',encoding='unicode_escape')
print("Sample Data")
print(df.head())
```

Sample Data

```
Company Name      Model Name Mobile Weight  RAM Front Camera \
0   Apple     iPhone 16 128GB    174g  6GB    12MP
1   Apple     iPhone 16 256GB    174g  6GB    12MP
2   Apple     iPhone 16 512GB    174g  6GB    12MP
3   Apple  iPhone 16 Plus 128GB  203g  6GB    12MP
4   Apple  iPhone 16 Plus 256GB  203g  6GB    12MP
```

```
Back Camera Processor Battery Capacity Screen Size \
0   48MP A17 Bionic  3,600mAh 6.1 inches
1   48MP A17 Bionic  3,600mAh 6.1 inches
2   48MP A17 Bionic  3,600mAh 6.1 inches
3   48MP A17 Bionic  4,200mAh 6.7 inches
4   48MP A17 Bionic  4,200mAh 6.7 inches
```

```
Launched Price (Pakistan) Launched Price (India) Launched Price (China) \
0      PKR 224,999      INR 79,999      CNY 5,799
1      PKR 234,999      INR 84,999      CNY 6,099
2      PKR 244,999      INR 89,999      CNY 6,499
3      PKR 249,999      INR 89,999      CNY 6,199
4      PKR 259,999      INR 94,999      CNY 6,499
```

```

Launched Price (USA) Launched Price (Dubai) Launched Year
0      USD 799        AED 2,799     2024
1      USD 849        AED 2,999     2024
2      USD 899        AED 3,199     2024
3      USD 899        AED 3,199     2024
4      USD 949        AED 3,399     2024
df={

'USN':[1,2,3,4,5],
'Name':['A','B','C','D','E'],
'Marks':[10,20,30,40,50]
}

df = pd.DataFrame(df)
print(df)

USN Name Marks
0  1   A   10
1  2   B   20
2  3   C   30
3  4   D   40
4  5   E   50
from sklearn.datasets import load_diabetes
diabetes= load_diabetes()
df=pd.DataFrame(diabetes.data,columns=diabetes.feature_names)
df['target']=diabetes.target
df.head()
print(df)

      age    sex    bmi    bp    s1    s2    s3 \
0  0.038076  0.050680  0.061696  0.021872 -0.044223 -0.034821 -0.043401
1  -0.001882 -0.044642 -0.051474 -0.026328 -0.008449 -0.019163  0.074412
2   0.085299  0.050680  0.044451 -0.005670 -0.045599 -0.034194 -0.032356
3  -0.089063 -0.044642 -0.011595 -0.036656  0.012191  0.024991 -0.036038
4   0.005383 -0.044642 -0.036385  0.021872  0.003935  0.015596  0.008142
..   ...
437 0.041708  0.050680  0.019662  0.059744 -0.005697 -0.002566 -0.028674
438 -0.005515  0.050680 -0.015906 -0.067642  0.049341  0.079165 -0.028674
439 0.041708  0.050680 -0.015906  0.017293 -0.037344 -0.013840 -0.024993
440 -0.045472 -0.044642  0.039062  0.001215  0.016318  0.015283 -0.028674
441 -0.045472 -0.044642 -0.073030 -0.081413  0.083740  0.027809  0.173816

      s4    s5    s6 target
0  -0.002592  0.019907 -0.017646  151.0
1  -0.039493 -0.068332 -0.092204   75.0
2  -0.002592  0.002861 -0.025930  141.0
3   0.034309  0.022688 -0.009362  206.0

```

```
4 -0.002592 -0.031988 -0.046641 135.0
.. ... ...
437 -0.002592 0.031193 0.007207 178.0
438 0.034309 -0.018114 0.044485 104.0
439 -0.011080 -0.046883 0.015491 132.0
440 0.026560 0.044529 -0.025930 220.0
441 -0.039493 -0.004222 0.003064 57.0
```

```
file_path='sample_sales_data.csv'
df=pd.read_csv(file_path)
print("Sample Data")
print(df.head())
```

Sample Data

	Product	Quantity	Price	Sales	Region
0	Laptop	5	1000	5000	North
1	Mouse	15	20	300	West
2	Keyboard	10	50	500	East
3	Monitor	8	200	1600	South
4	Laptop	12	950	11400	North

In [30]:

```
df=pd.read_csv('/content/Dataset of Diabetes .csv')
print("Sample Data")
print(df.head())
```

Sample Data

	ID	No_Pation	Gender	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	\
0	502	17975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	
1	735	34221	M	26	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	
2	420	47975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	
3	680	87656	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	
4	504	34223	M	33	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	

BMI CLASS

0	24.0	N
1	23.0	N
2	24.0	N
3	24.0	N
4	21.0	N

In [28]:

```
df.to_csv('output.csv', index=False)
print("Data saved to output.csv")
```

Data saved to output.csv

```
sales_df = pd.read_csv('sample_sales_data.csv')
sales_by_region = sales_df.groupby('Region')[['Sales']].sum()
print("\nTotal sales by region:")
print(sales_by_region)
best_selling_products = sales_df.groupby('Product')[['Quantity']].sum().sort_values(ascending=False)
print("\nBest-selling products by quantity:")
print(best_selling_products)
# Saving the sales by region data to a CSV file
sales_by_region.to_csv('sales_by_region.csv')
# Saving the best-selling products data to a CSV file
best_selling_products.to_csv('best_selling_products.csv')
print("\nAnalysis results saved to CSV files.")
```

Total sales by region:

Region	Sales
East	770
North	16400
South	3070
West	650

Name: Sales, dtype: int64

Best-selling products by quantity:

Product	Quantity
Mouse	29
Laptop	17
Keyboard	16
Monitor	15

Name: Quantity, dtype: int64

Analysis results saved to CSV files.

```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
```

Step 2: Downloading Stock Market Data

Define the ticker symbols for Indian companies

Example: Reliance Industries (RELIANCE.NS), TCS (TCS.NS), Infosys (INFY.NS)

tickers = ["RELIANCE.NS", "TCS.NS", "INFY.NS"]

Fetch historical data for the last 1 year

```
data = yf.download(tickers, start="2022-10-01", end="2023-10-01", group_by='ticker')
```

```
# Display the first 5 rows of the dataset
```

```
print("First 5 rows of the dataset:")
```

```
print(data.head())
```

```
YF.download() has changed argument auto_adjust default to True
```

```
[*****100%*****] 3 of 3 completed
```

```
First 5 rows of the dataset:
```

Ticker	RELIANCE.NS	\\			
Price	Open	High	Low	Close	Volume
Date					
2022-10-03	1088.493134	1100.076669	1075.521371	1078.479858	11852723
2022-10-04	1091.360592	1100.554607	1087.878645	1098.369873	8948850
2022-10-06	1105.561293	1115.119370	1100.622856	1102.420654	13352162
2022-10-07	1099.029980	1112.343173	1099.029980	1107.086182	7714340
2022-10-10	1094.637648	1100.372591	1086.900123	1095.001831	6329527

Ticker	TCS.NS	\\			
Price	Open	High	Low	Close	Volume
Date					
2022-10-03	2799.043864	2823.062325	2779.417847	2789.651367	1763331
2022-10-04	2831.707536	2895.305232	2825.212304	2888.903320	2145875
2022-10-06	2907.454507	2919.603947	2890.118145	2898.996582	1790816
2022-10-07	2894.744206	2901.847047	2858.015696	2864.370605	1939879
2022-10-10	2813.062865	2922.407833	2808.390003	2914.510742	3064063

Ticker	INFY.NS	\\			
Price	Open	High	Low	Close	Volume
Date					
2022-10-03	1337.743240	1337.743240	1313.110574	1320.453003	4943169
2022-10-04	1345.038201	1356.928245	1339.638009	1354.228149	6631341
2022-10-06	1369.007786	1383.029504	1368.155094	1378.624023	6180672
2022-10-07	1370.286676	1381.181893	1364.412779	1374.881592	3994466
2022-10-10	1351.338576	1387.956005	1351.338576	1385.729614	5274677

```
In [38]:
```

```
# Step 3: Basic Data Exploration
```

```
# Check the shape of the dataset
```

```
print("\nShape of the dataset:")
```

```
print(data.shape)
```

```
# Check column names
```

```
print("\nColumn names:")
```

```
print(data.columns)
```

```
# Summary statistics for a specific stock (e.g., Reliance)
```

```

reliance_data = data['RELIANCE.NS']
print("\nSummary statistics for Reliance Industries:")
print(reliance_data.describe())

```

Shape of the dataset:

(247, 15)

Column names:

```

MultiIndex([('RELIANCE.NS', 'Open'),
            ('RELIANCE.NS', 'High'),
            ('RELIANCE.NS', 'Low'),
            ('RELIANCE.NS', 'Close'),
            ('RELIANCE.NS', 'Volume'),
            ('TCS.NS', 'Open'),
            ('TCS.NS', 'High'),
            ('TCS.NS', 'Low'),
            ('TCS.NS', 'Close'),
            ('TCS.NS', 'Volume'),
            ('INFY.NS', 'Open'),
            ('INFY.NS', 'High'),
            ('INFY.NS', 'Low'),
            ('INFY.NS', 'Close'),
            ('INFY.NS', 'Volume')],
           names=['Ticker', 'Price'])

```

Summary statistics for Reliance Industries:

	Open	High	Low	Close	Volume
count	247.000000	247.000000	247.000000	247.000000	2.470000e+02
mean	1147.548565	1156.216094	1137.195377	1146.522438	1.316652e+07
std	65.890232	66.850140	65.752477	66.687266	6.754099e+06
min	1008.159038	1010.434788	992.228728	1001.900696	3.370033e+06
25%	1098.881943	1103.399363	1084.795104	1097.357178	8.717141e+06
50%	1147.435126	1155.036230	1138.787343	1147.253052	1.158959e+07
75%	1196.261444	1203.625226	1184.985050	1195.064575	1.530302e+07
max	1288.076823	1299.910771	1273.056848	1293.470337	5.708188e+07

In [39]:

```

# Calculate daily returns
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
print("\nDaily returns for Reliance Industries:")
print(reliance_data['Daily Return'].head())

```

#

Daily returns for Reliance Industries:

Date

```
2022-10-03      NaN
2022-10-04  0.018443
2022-10-06  0.003688
2022-10-07  0.004232
2022-10-10 -0.010915
```

Name: Daily Return, dtype: float64

<ipython-input-39-2bd40fbe735c>:2: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

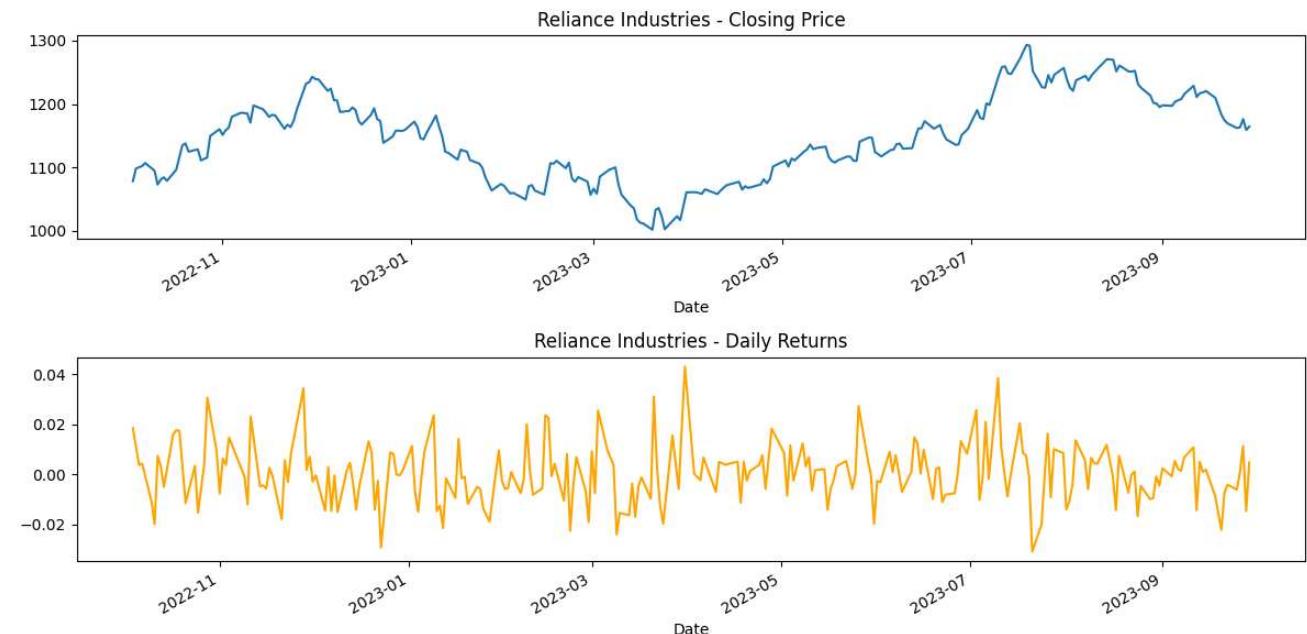
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
```

In [40]:

```
# Plot the closing price and daily returns
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
reliance_data['Close'].plot(title="Reliance Industries - Closing Price")
plt.subplot(2, 1, 2)
reliance_data['Daily Return'].plot(title="Reliance Industries - Daily Returns", color='orange')
plt.tight_layout()
plt.show()
```



In [41]:

```
# Step 4: Saving the Processed Data to a New CSV File
```

```
# Save the Reliance data to a CSV file
reliance_data.to_csv('reliance_stock_data.csv')
print("\nReliance stock data saved to 'reliance_stock_data.csv'.
```

Reliance stock data saved to 'reliance_stock_data.csv'.

In [43]:

```
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
data = yf.download(tickers, start="2024-01-01", end="2024-12-30", group_by='ticker')
```

[*****100%*****] 3 of 3 completed

In [44]:

```
hdfc_data = data['HDFCBANK.NS']
hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()
print("\nDaily returns for HDFC Industries:")
print(hdfc_data['Daily Return'].head())
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
hdfc_data['Close'].plot(title="HDFC Industries - Closing Price")
plt.subplot(2, 1, 2)
hdfc_data['Daily Return'].plot(title="HDFC Industries - Daily Returns", color='orange')
plt.tight_layout()
plt.show()
```

```
<ipython-input-44-f8f2798a87f5>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

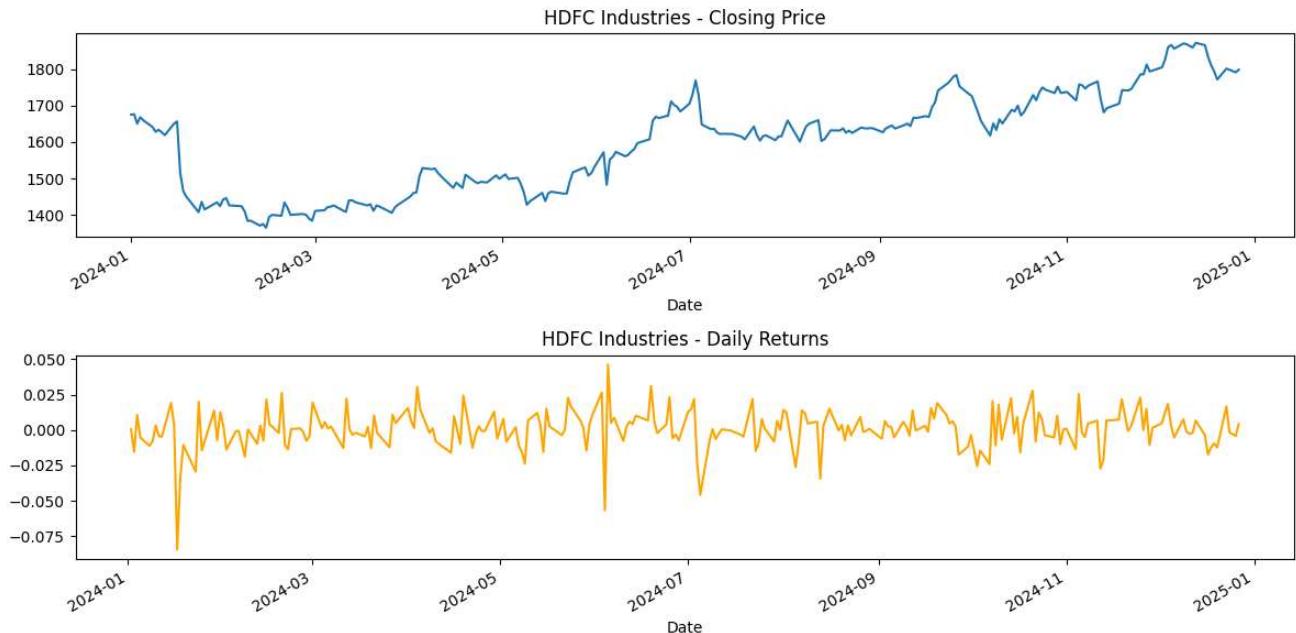
```
hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()
```

Daily returns for HDFC Industries:

Date

2024-01-01	NaN
2024-01-02	0.000589
2024-01-03	-0.015420
2024-01-04	0.010730
2024-01-05	-0.005116

Name: Daily Return, dtype: float64



In [45]:

```
icici_data = data['ICICIBANK.NS']
icici_data['Daily Return'] = icici_data['Close'].pct_change()
print("\nDaily returns for ICICI Industries:")
print(icici_data['Daily Return'].head())
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
icici_data['Close'].plot(title="ICICI Industries - Closing Price")
plt.subplot(2, 1, 2)
icici_data['Daily Return'].plot(title="ICICI Industries - Daily Returns", color='orange')
plt.tight_layout()
plt.show()
```

<ipython-input-45-6f81270ab07d>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

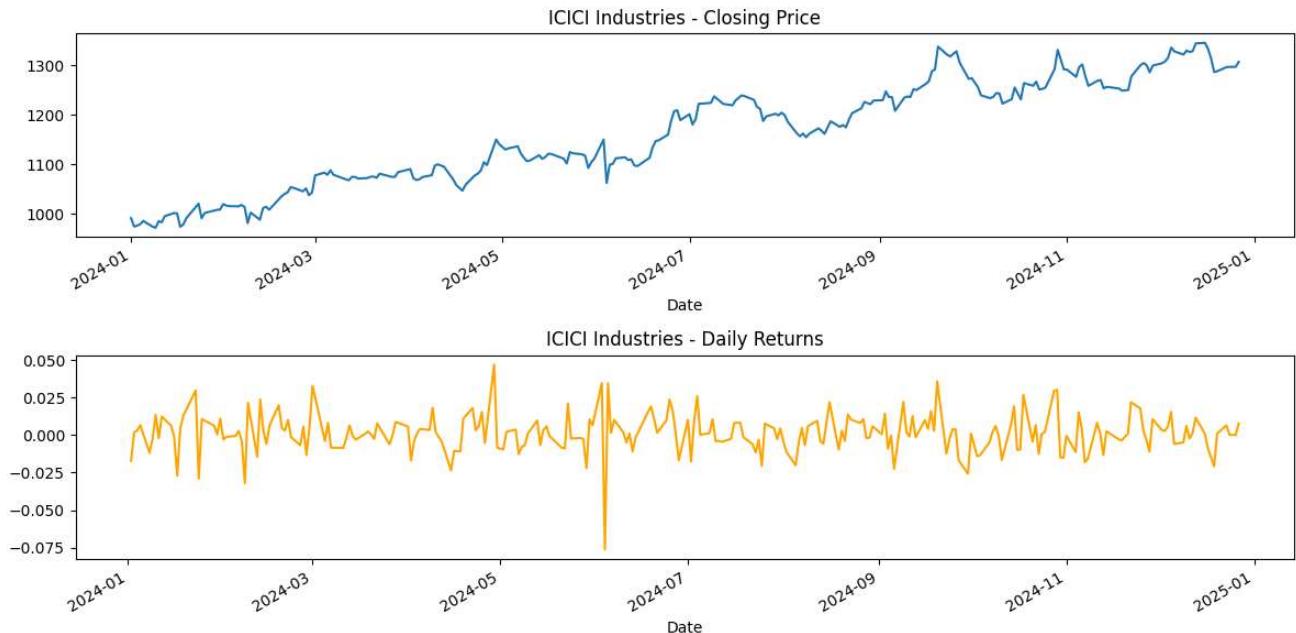
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
icici_data['Daily Return'] = icici_data['Close'].pct_change()
```

Daily returns for ICICI Industries:

Date	Daily Return
2024-01-01	NaN
2024-01-02	-0.017160
2024-01-03	0.001833
2024-01-04	0.003150
2024-01-05	0.006635

Name: Daily Return, dtype: float64



In [46]:

```
kotak_data = data['KOTAKBANK.NS']
kotak_data['Daily Return'] = kotak_data['Close'].pct_change()
print("\nDaily returns for Kotak Industries:")
print(kotak_data['Daily Return'].head())
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
kotak_data['Close'].plot(title="Kotak Industries - Closing Price")
plt.subplot(2, 1, 2)
kotak_data['Daily Return'].plot(title="Kotak Industries - Daily Returns", color='orange')
plt.tight_layout()
plt.show()
```

<ipython-input-46-098de9be386a>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

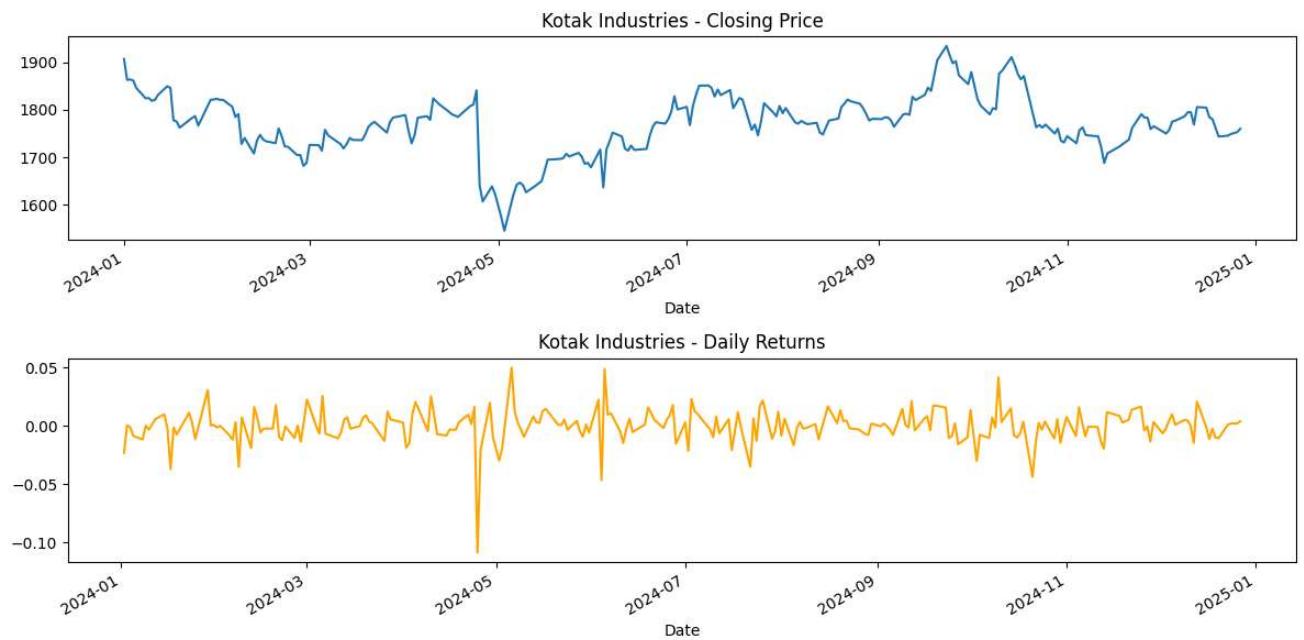
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

kotak_data['Daily Return'] = kotak_data['Close'].pct_change()

Daily returns for Kotak Industries:

Date	Daily Return
2024-01-01	NaN
2024-01-02	-0.023099
2024-01-03	0.000456
2024-01-04	-0.001233
2024-01-05	-0.008586

Name: Daily Return, dtype: float64



Program 3

Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.

Algorithm:

Instance	a_2	a_3	Class
1	Hot	High	No
2	Hot	High	No
3	Cool	High	No
4	Hot	High	No
5	Hot	Normal	Yes

$$\text{Entropy}(S) = -\frac{4}{5} \log_2 \left(\frac{4}{5}\right) - \frac{1}{5} \log_2 \left(\frac{1}{5}\right)$$

for a_2

$$S_{\text{hot}} [1+, 3-] = -\frac{1}{4} \log \left(\frac{1}{4}\right) - \frac{3}{4} \log \left(\frac{3}{4}\right)$$

$$= 0.7219$$

~~$$S_{\text{cool}} [0+, 3-] = -\frac{1}{4} \log \left(\frac{1}{4}\right) - \frac{3}{4} \log \left(\frac{3}{4}\right)$$~~

~~$$S_{\text{cool}} [0+, 1-] = 0$$~~

$$\text{Gain}(S, a_2) = 0.7219 - \frac{4}{5} (0.7219) - 0 = 0$$

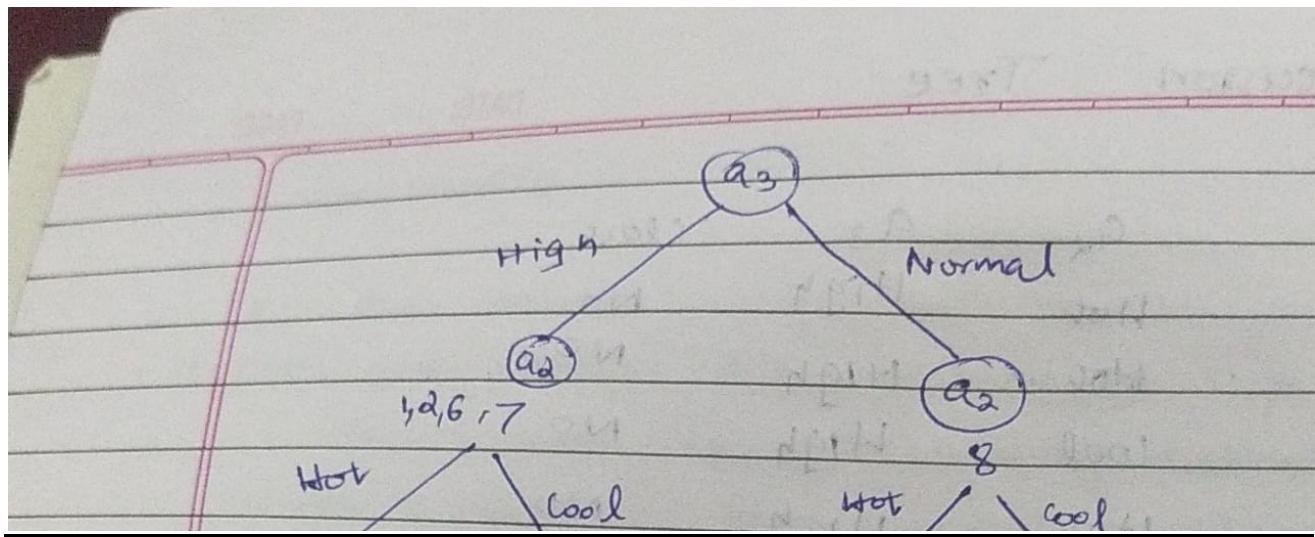
For a_3

~~$$S_{\text{high}} = [0+, 4-] = 0$$~~

~~$$S_{\text{normal}} = [1+, 0-] = 0$$~~

$$\text{Gain}(S, a_3) = 0.7219 - 0 - 0 = 0.7219$$

$\therefore a_3$ is the next node since



Decision Tree classifier

1 Iris dataset:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
df = pd.read_csv('content/iris.csv')
print(df.head(1))
```

```
X-train, X-test, Y-train, Y-test = train_test_split(df,
→ random_state=1), df['species'], test_size=0.2, random_state
clf = DecisionTreeRegressor(criterion='entropy')
clf.fit(X-train, Y-train)
```

```
Y-pred = clf.predict(X-test)
```

```
accuracy = accuracy_score(Y-test, Y-pred)
```

```
print(f"Accuracy: {accuracy:.2f}%")
```

```
conf = confusion_matrix(Y-test, Y-pred)
```

```
print("Confusion Matrix: ")
```

```
print(conf)
```

```
print(classification_report(Y-test, Y-pred))
```

```
plt.figure(figsize=(12,8))
```

```
plot_tree(clf, filled=True, feature_names=X-train)
plt.show()
```

Output:

	sepal length	sepal width	petal length	petal width
0	5.1	3.5	1.4	0.2

DATE:

	Precision	recall	f1-score	supp
Iris-setosa	1.00	1.00	1.00	10
Iris-versicolor	1.00	1.00	1.00	9
Iris-Virginica	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

There were no misclassifications.

The confusion matrix tells us that the model perfectly on the predict training datasets.

2. Petrol consumption dataset

what the confusion matrix tell you about the model's performance? were there

can you interpret the regression tree structure
what are the most important features for prediction of petrol consumption

2 The tree structure would show if Pet then go left else go right

if Average-income < 24.06 then predict 714
predict 603

Most important features:

Average-income, petroletax, population-driver-licenc

Code:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Create the dataset
data = {
    'a1': [True, True, False, False, True, True, True, False, False],
    'a2': ['Hot', 'Hot', 'Hot', 'Cool', 'Cool', 'Cool', 'Hot', 'Hot', 'Cool'],
    'a3': ['High', 'High', 'High', 'Normal', 'Normal', 'High', 'High', 'Normal', 'Normal'],
    'Classification': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes']
}

data

# Convert to DataFrame
df = pd.DataFrame(data)

# Convert categorical data to numerical data
label_encoders = {}
for column in df.columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

# Split the dataset into features and target
X = df.drop('Classification', axis=1)
y = df['Classification']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize the Decision Tree Classifier with entropy as the criterion
clf = DecisionTreeClassifier(criterion='entropy')

# Train the classifier
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)
```

```

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test, y_pred, target_names=['No', 'Yes']))

# Optionally, visualize the decision tree
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(12,8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=['No', 'Yes'])
plt.show()

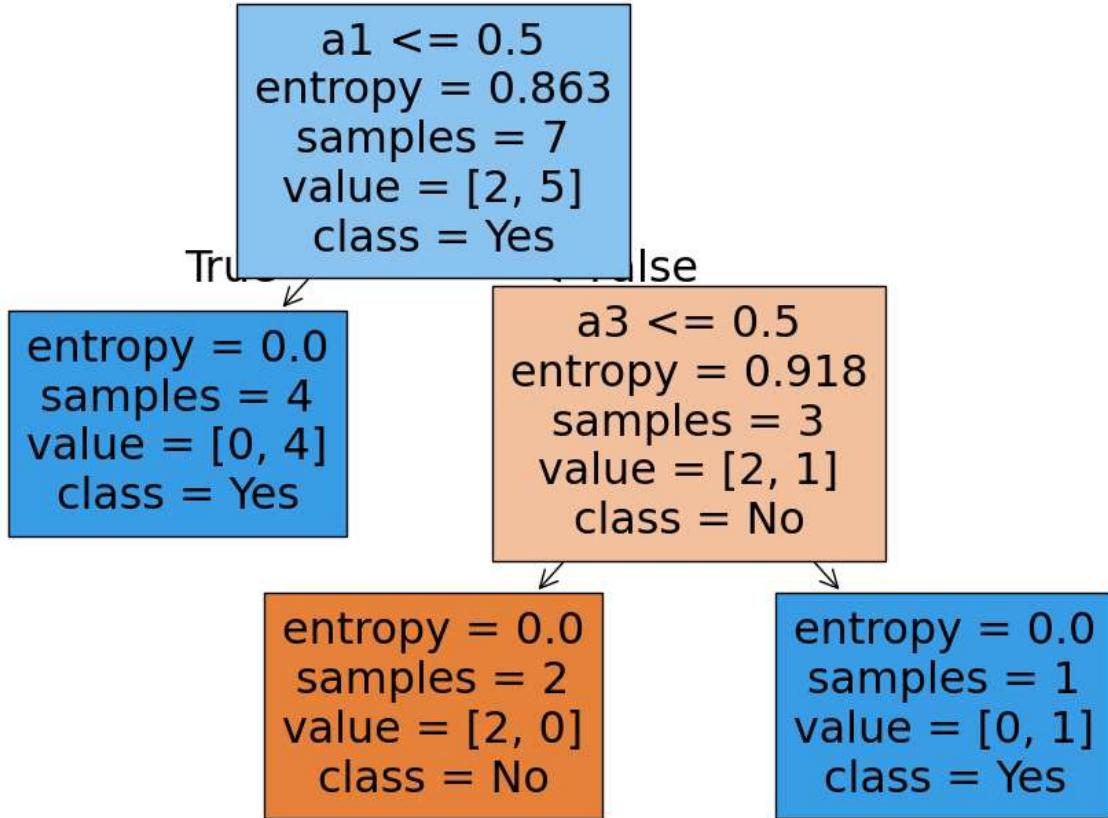
```

Accuracy: 1.00

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

No	1.00	1.00	1.00	2
Yes	1.00	1.00	1.00	1

accuracy		1.00	3	
macro avg	1.00	1.00	1.00	3
weighted avg	1.00	1.00	1.00	3



```

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load the iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the model

```

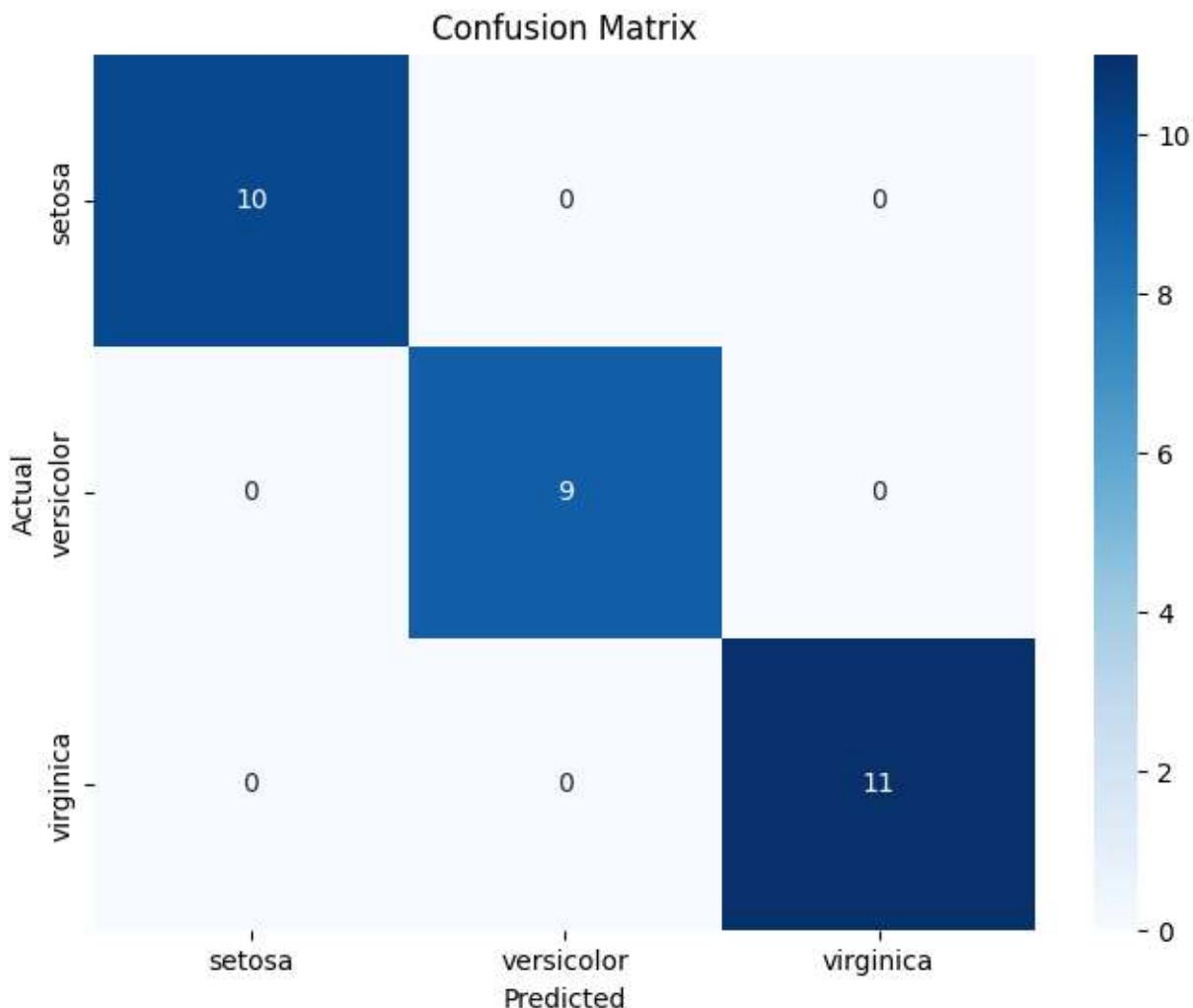
```

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Create and display the confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

```

Accuracy: 1.0



```

import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

```

```

import seaborn as sns
import matplotlib.pyplot as plt

try:
    df = pd.read_csv('drug.csv')
except FileNotFoundError:
    print("Error: 'drugdataset.csv' not found. Please upload the file or provide the correct path.")
    exit()

# Identify categorical columns (replace with your actual column names)
categorical_cols = ['Sex', 'BP', 'Cholesterol', 'Na_to_K']

# Use one-hot encoding for categorical features
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)

# Split data into features (X) and target (y)
X = df.drop('Drug', axis=1) # Assuming 'Drug' is the target variable
y = df['Drug']

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the DecisionTreeClassifier
clf = DecisionTreeClassifier()

# Train the classifier
clf.fit(X_train, y_train)

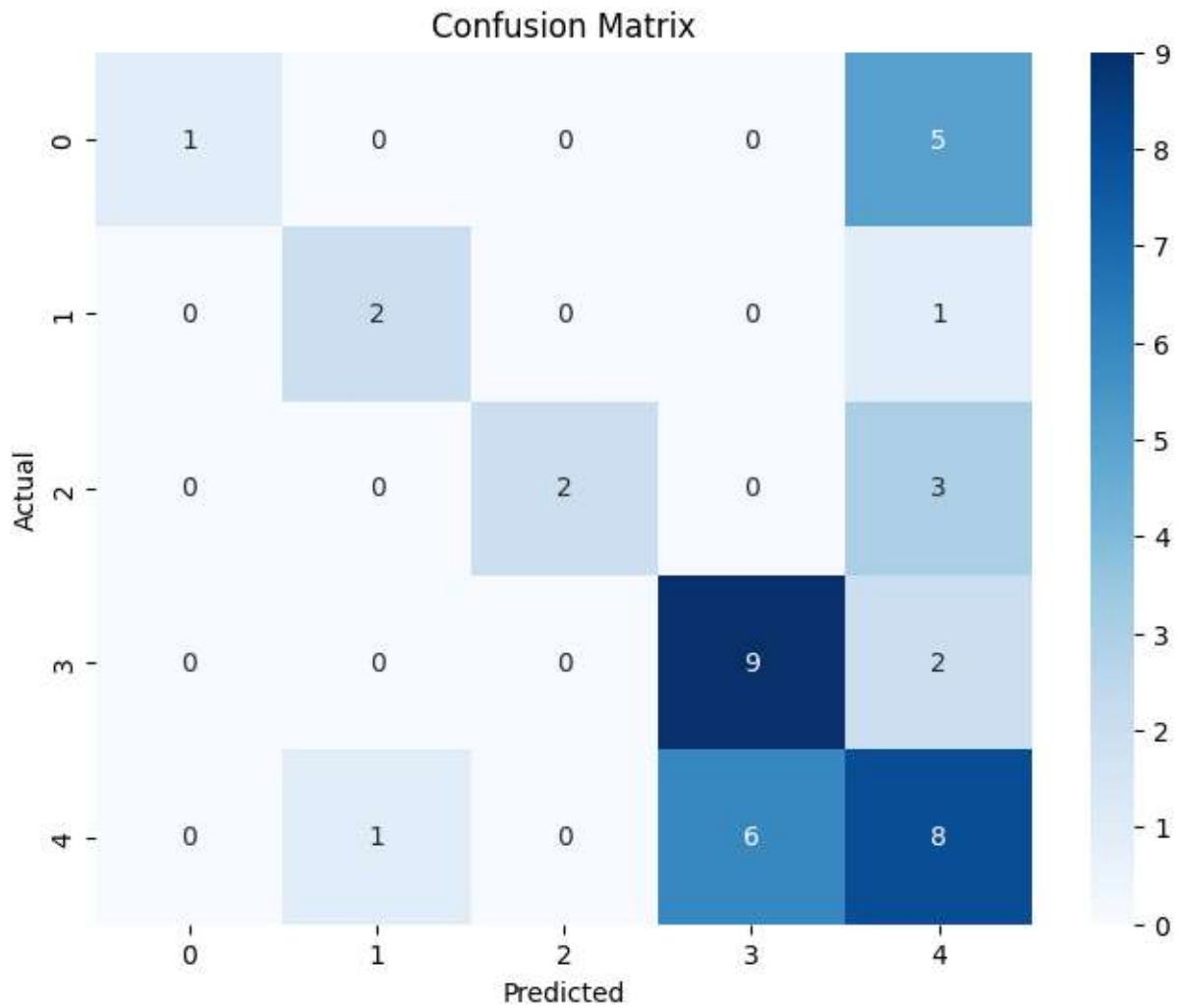
# Make predictions on the test data
y_pred = clf.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')

# Create and display the confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

```

Accuracy: 0.55



```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error

df = pd.read_csv('petrol_consumption.csv')

# Split data into features (X) and target (y)
X = df.drop('Petrol_Consumption', axis=1)
y = df['Petrol_Consumption']

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the DecisionTreeRegressor
regressor = DecisionTreeRegressor()

# Train the regressor

```

```
regressor.fit(X_train, y_train)

# Make predictions on the test data
y_pred = regressor.predict(X_test)

# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mse**0.5

print(f'Mean Absolute Error: {mae}')
print(f'Mean Squared Error: {mse}')
print(f'Root Mean Squared Error: {rmse}')

Mean Absolute Error: 96.2
Mean Squared Error: 17858.2
Root Mean Squared Error: 133.63457636405332
```

Program 4

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Algorithm:

LAB - 4

DATE: 11/31/3

Implement Linear Regression and multiple linear regression

x_i (week)	y_i (sales in thousands)
1	2
2	4
3	5
4	9

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}$$

$$x^T = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$

$$Y = \begin{bmatrix} 2 \\ 4 \\ 5 \\ 9 \end{bmatrix}$$

$$x^T x = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix} \Rightarrow x^T x^{-1} = \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.3 \end{bmatrix}$$

$$(x^T x)^{-1} x^T = Y$$

$$\Rightarrow \boxed{\begin{bmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.3 & -0.1 & 0.1 & 0.3 \end{bmatrix}}$$

$$\beta = ((x^T x)^{-1} x^T) Y$$

$$\beta = \begin{bmatrix} -0.5 \\ 2.2 \end{bmatrix}$$

$$y_i = -0.5 + 2.2x_i$$

Q2 Linear regression

```

df = pd.read_csv('content/canada-per capita income.csv')
reg = fit(df[['Year']], df['per capita income (US$)'])
print("a")
import pandas as pd
from sklearn.linear_model import LinearRegression
df = pd.read_csv('content/canada.csv')
reg = LinearRegression()
reg.fit(df[['Year']], df['Salary'])
print("a0:", reg.intercept_)
print("a1:", reg.coef_)

```

Output for 1: -1640698.9436 2020 → 4832.7982

Output for 2: -46425.2487 12 → 12331
6409.1303

Q3 Multiple linear regression

```

import pandas as pd
df = pd.read_csv('content/hiring.csv')
df = df.fillna(0)
reg3 = LinearRegression()
reg3 = LinearRegression()
reg3.fit(df[['experience', 'test score(out of 10)', 'interview']]]

```

Output 1:

$$a_0 = 75920.8633$$

$$[a_1, a_2, a_3] = [0, 659.4789, -1990.4076]$$

$$\text{prediction 1: } 69913.6690$$

$$\text{Output 2: prediction 2: } 62611.5107$$

Output 2:

$$a_{0b} = -82439.1556$$

$$[a_1, a_2, a_3, a_4] = [5.3364 e-01, 1.1389 e+00, 8.2167 e-01, -8.7449 e+02]$$

$$\text{Prediction: } 54066.30$$

Code:

```

import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt

df = pd.read_csv('canada_per_capita_income.csv')
df

# Commented out IPython magic to ensure Python compatibility.
# %matplotlib inline
plt.xlabel('year')
plt.ylabel('per capita income (US$)')
plt.scatter(df['year'], df['per capita income (US$)'], color='red', marker='+')

new_df = df.drop('per capita income (US$)', axis='columns')
new_df

price = df['per capita income (US$)']

# Create linear regression object
reg = linear_model.LinearRegression()
reg.fit(new_df, price)

print(reg.predict([[2020]]))

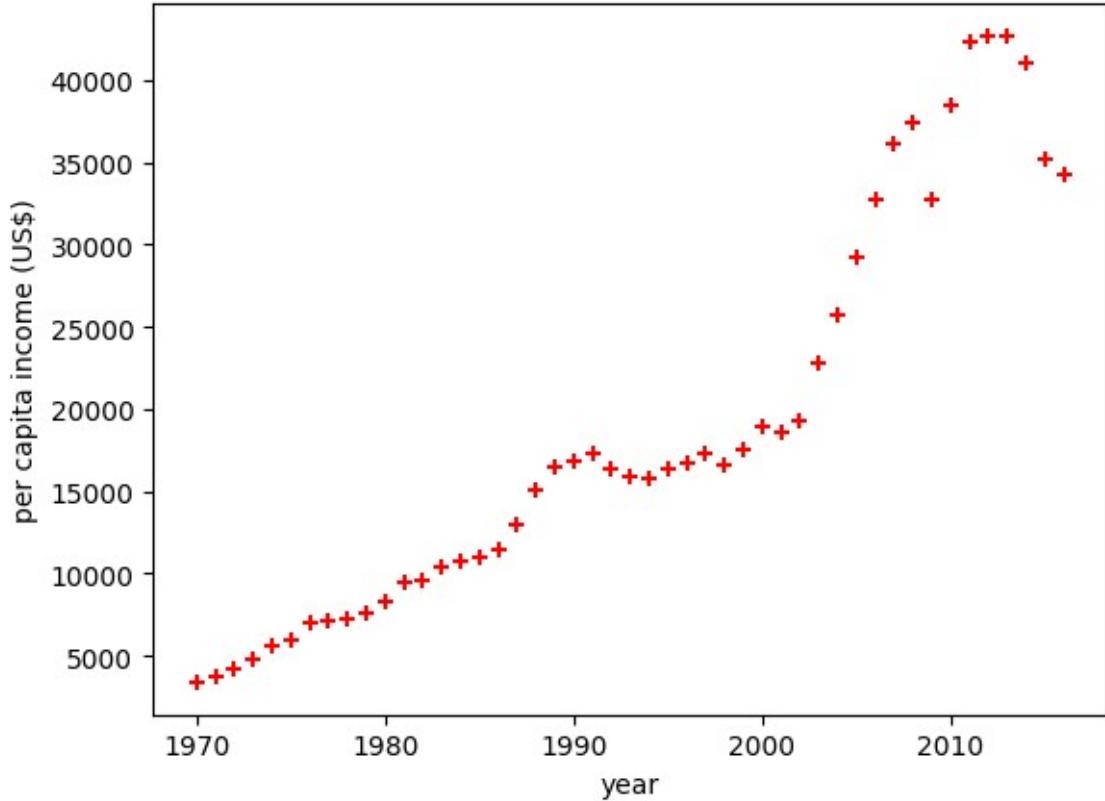
print(reg.coef_)

print(reg.intercept_)

"""Y = m * X + b (m is coefficient and b is intercept)"""

[41288.69409442]
[828.46507522]
-1632210.7578554575
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not
have valid feature names, but LinearRegression was fitted with feature names
    warnings.warn(
Out[31]:
'Y = m * X + b (m is coefficient and b is intercept)'

```



In [15]:

```
# -*- coding: utf-8 -*-
#2. LINEAR_LR
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt

df = pd.read_csv('salary.csv')
df

# Commented out IPython magic to ensure Python compatibility.
# %matplotlib inline
plt.xlabel('YearsExperience')
plt.ylabel('Salary')
plt.scatter(df['YearsExperience'], df['Salary'], color='red', marker='+')

df['YearsExperience']=df['YearsExperience'].fillna(df['YearsExperience'].mean())
new_df = df.drop('Salary', axis='columns')
new_df

salary = df['Salary']

# Create linear regression object
```

```

reg = linear_model.LinearRegression()
reg.fit(new_df,salary)

reg.predict([[12]])

print(reg.coef_)

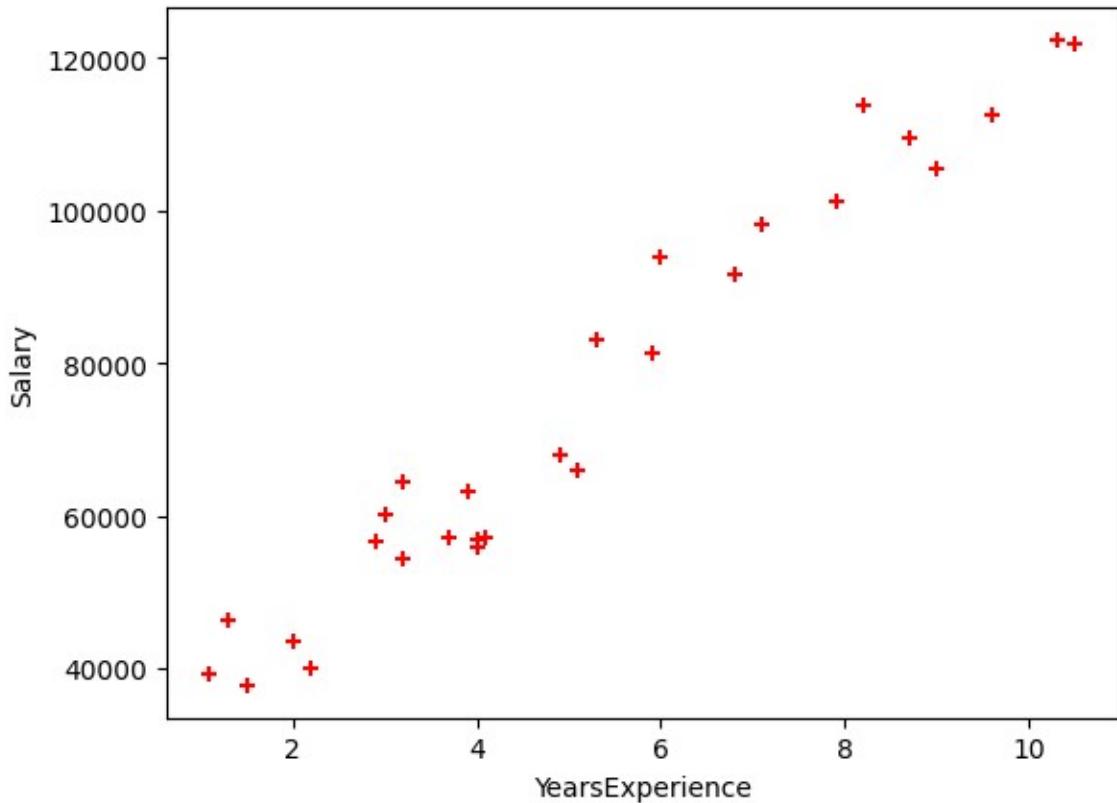
print(reg.intercept_)

"""Y = m * X + b (m is coefficient and b is intercept)"""

print(reg.predict([[12]]))

[9398.64060184]
27197.2020175696
[139980.88923969]
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not
have valid feature names, but LinearRegression was fitted with feature names
    warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not
have valid feature names, but LinearRegression was fitted with feature names
    warnings.warn(

```



In [17]:

```

!pip install word2number

Collecting word2number
  Downloading word2number-1.1.zip (9.7 kB)
    Preparing metadata (setup.py) ... done
Building wheels for collected packages: word2number
  Building wheel for word2number (setup.py) ... done
    Created wheel for word2number: filename=word2number-1.1-py3-none-any.whl size=5568
sha256=88e1bcbd080599c701a9fbaf1d8af7e1c4b020a078bd8566ebf1efa7804baf89
  Stored in directory:
  /root/.cache/pip/wheels/cd/ef/ae/073b491b14d25e2efafcffca9e16b2ee6d114ec5c643ba4f06
Successfully built word2number
Installing collected packages: word2number
Successfully installed word2number-1.1
In [20]:

```

```

#1 MULTIPLE_LR
# -*- coding: utf-8 -*-
from word2number import w2n
import pandas as pd
import numpy as np
from sklearn import linear_model

df = pd.read_csv('hiring.csv')
df

"""Data Preprocessing: Fill NA values with median value of a column"""

df.experience = df.experience.bfill()
# df.experience = df.experience.fillna("zero")
# df

df.experience = df.experience.apply(w2n.word_to_num)
# df
df['test_score(out of 10)'] = df['test_score(out of 10)'].fillna(df['test_score(out of 10)'].mean())
# df

reg = linear_model.LinearRegression()
reg.fit(df.drop('salary($)',axis='columns'),df['salary($)'])

reg.coef_

reg.intercept_

print(reg.predict([[2, 9, 6]]))
print(reg.predict([[12, 10, 10]]))

```

```
[45968.51077949]
[92712.4732389]
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not
have valid feature names, but LinearRegression was fitted with feature names
    warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not
have valid feature names, but LinearRegression was fitted with feature names
    warnings.warn(
In [32]:
```

```
#2. MULTIPLE_LR
import pandas as pd
import numpy as np
from sklearn import linear_model

df = pd.read_csv('1000_Companies.csv')

from sklearn.preprocessing import OrdinalEncoder
ord_enc = OrdinalEncoder()
df["State"] = ord_enc.fit_transform(df[["State"]])

reg = linear_model.LinearRegression()
reg.fit(df.drop('Profit',axis='columns'),df['Profit'])

print(reg.coef_)

print(reg.intercept_)

print(reg.predict([[91694.48, 515841.3, 11931.24, 1.0]]))

[ 0.55389023  1.02672443  0.08058525 46.62387015]
-70214.44175560221
[511209.20332292]
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not
have valid feature names, but LinearRegression was fitted with feature names
    warnings.warn(
In [28]:
```

```
pd.get_dummies(df, columns=["State"]).head()
from sklearn.preprocessing import OrdinalEncoder
ord_enc = OrdinalEncoder()
df["State"] = ord_enc.fit_transform(df[["State"]])
df.head()
```

```
Out[28]:
```

	R&D Spend	Administration	Marketing Spend	State	Profit	state_code
0	165349.20	136897.80	471784.10	2.0	192261.83	2.0
1	162597.70	151377.59	443898.53	0.0	191792.06	0.0
2	153441.51	101145.55	407934.54	1.0	191050.39	1.0
3	144372.41	118671.85	383199.62	2.0	182901.99	2.0
4	142107.34	91391.77	366168.42	1.0	166187.94	1.0

```

import pandas as pd
import numpy as np
from sklearn import linear_model

df = pd.read_csv('homeprices_Multiple_LR.csv')
df

"""Data Preprocessing: Fill NA values with median value of a column"""

df.bedrooms.median()

df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())
df

reg = linear_model.LinearRegression()
reg.fit(df.drop('price',axis='columns'),df.price)

reg.coef_
reg.intercept_

"""Find price of home with 3000 sqr ft area, 3 bedrooms, 40 year old"""

reg.predict([[3000, 3, 40]])

112.06244194*3000 + 23388.88007794*3 + -3231.71790863*40 + 221323.00186540384

498408.25157402386

import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt

```

```

df = pd.read_csv('housing_area_price.csv')
df

# Commented out IPython magic to ensure Python compatibility.
# %matplotlib inline
plt.xlabel('area')
plt.ylabel('price')
plt.scatter(df.area, df.price, color='red', marker='+')

new_df = df.drop('price', axis='columns')
new_df

price = df.price
price

# Create linear regression object
reg = linear_model.LinearRegression()
reg.fit(new_df, price)

"""(1) Predict price of a home with area = 3300 sqr ft"""

reg.predict([[3300]])

reg.coef_

reg.intercept_

"""Y = m * X + b (m is coefficient and b is intercept)"""

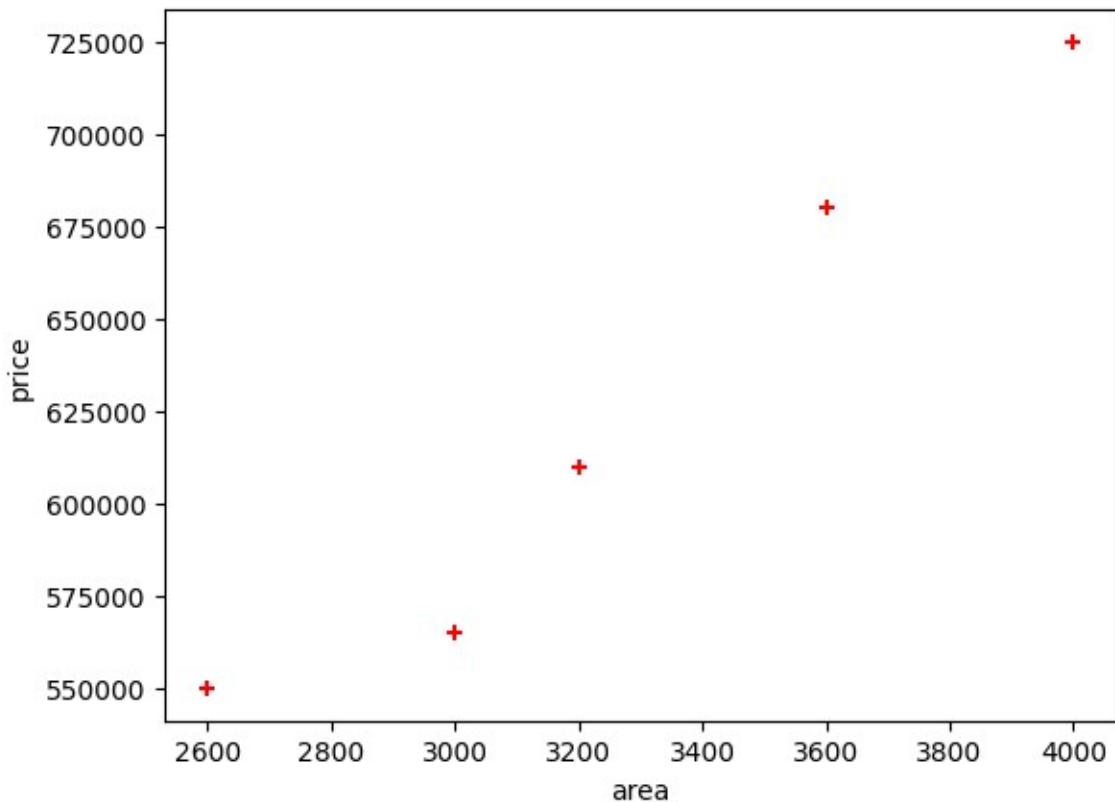
3300*135.78767123 + 180616.43835616432

"""(1) Predict price of a home with area = 5000 sqr ft"""

reg.predict([[5000]])

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not
have valid feature names, but LinearRegression was fitted with feature names
    warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not
have valid feature names, but LinearRegression was fitted with feature names
    warnings.warn(
Out[4]:
array([859554.79452055])

```



In []:

Program 5

Build Logistic Regression Model for a given dataset

Algorithm:

Lab - 3

DATE: 12/3/25 P.

$$\textcircled{1} \quad a_0 = -5 \quad a_1 = 0.8$$

(i) Logistic regression \rightarrow sigmoid

$$p(x) = \frac{1}{1 + e^{-(a_0 + a_1 x)}} = \frac{1}{1 + e^{(-5 + 0.8x)}}$$

(ii) calculate the probability that a student who studies 7 hours will pass

$$x = 7 \rightarrow p(x) = \frac{1}{1 + e^{-(5 + 0.8(7))}} = 0.6457$$

(iii) Determine the predicted class (pass/fail) for this based on threshold of 0.5

$$p(x) = 0.6457$$

$$p(x) \geq 0.5$$

Thus $y=1$ (pass)

(a) consider $z = [2, 1, 0]$ for 3 classes. Apply softmax for this student to find probability value classes

$$\text{softmax}(z_k) = \frac{e^{z_k}}{\sum_{j=1}^n e^{z_j}}$$

$$\text{softmax}(z_1) = \frac{e^2}{e^2 + e^1 + e^0} = 0.665$$

$$\text{softmax}(z_2) = \frac{e^1}{e^2 + e^1 + e^0} = 0.244$$

Binary Logistic Regression

```
import pandas as pd  
from matplotlib.pyplot import pyplot as plt  
df = pd.read_csv("insurance_data.csv")  
df.head()  
from sklearn.model_selection import train_test_split  
x_train, x-test, y-train, y-test = train_test_split (df[["  
train size = 0.8])  
from sklearn.linear_model import LogisticRegression  
model = LogisticRegression()  
model.fit (x-train, y-train)  
y-pred = model.predict (x-test)  
model.score (x-test, y-test)  
model.predict_proba (x-test)  
y-pred = model.predict ([[60]])  
print ("Model co-efficients: ", model.coef_ )  
print ("Model intercept: ", model.intercept_ )  
import math  
def sigmoid(x):  
    return 1 / (1 + math.exp(-x))  
  
def pred_func (age):  
    z = 0.127 * age - 4.973  
    y = sigmoid(z)  
    return y
```

significantly.

For Zoo dataset

- (i) Did you perform data preprocessing steps? If
- Yes, it has steps like merging datasets, h
values & encoding ~~int~~ categorical variables to
data for logistic regression.
- (ii) Were there missing values? How to handle them?
- No missing values are present.
- (iii) What does confusion matrix tell?
- It reveals how well the model predicts
types and misclassification often occurred.
- (iv) Which class types were most frequently m
Misclassification often occurs due to over

DATE: 18/11/

```
Multi-class Logistic Regression  
from sklearn.datasets import load_iris  
from sklearn.model_selection import train_test_split  
from sklearn.linear_model import LogisticRegression  
from sklearn.metrics import accuracy_score  
import matplotlib.pyplot as plt  
iris = pd.read_csv("iris.csv")  
X = iris.drop("species", axis="columns")  
y = iris.species  
X_train, X_test, y_train, y_test = train_test_split(X, y)  
model = LogisticRegression(multi_class='multinomial')  
model.fit(X_train, y_train)  
y_pred = model.predict(X_test)  
acc = accuracy_score(y_test, y_pred)  
print(f"Accuracy of model = {accuracy=:.2f}")  
conf_matrix = metrics.confusion_matrix(y_test, y_pred)  
print("Confusion matrix", conf_matrix)
```

Output: Accuracy of the model = 1.00

Confusion matrix:

$$\begin{bmatrix} 10 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 11 \end{bmatrix}$$

For HR Dataset:

- (i) Which variable has a direct & clear impact

Code:

```
df = pd.read_csv("insurance_data.csv")
df.head()

plt.scatter(df.age, df.bought_insurance, marker='+', color='red')

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(df[['age']], df.bought_insurance, train_size=0.9, random_state=10)
X_train.shape

X_test

from sklearn.linear_model import LogisticRegression
model = LogisticRegression()

model.fit(X_train, y_train)

X_test

y_test

y_predicted = model.predict(X_test)
y_predicted

model.score(X_test, y_test)

model.predict_proba(X_test)

y_predicted = model.predict([[60]])
y_predicted

#model.coef_ indicates value of m in y=m*x + b equation
model.coef_

#model.intercept_ indicates value of b in y=m*x + b equation
model.intercept_

#Lets defined sigmoid function now and do the math with hand
import math
def sigmoid(x):
    return 1 / (1 + math.exp(-x))

def prediction_function(age):
```

```

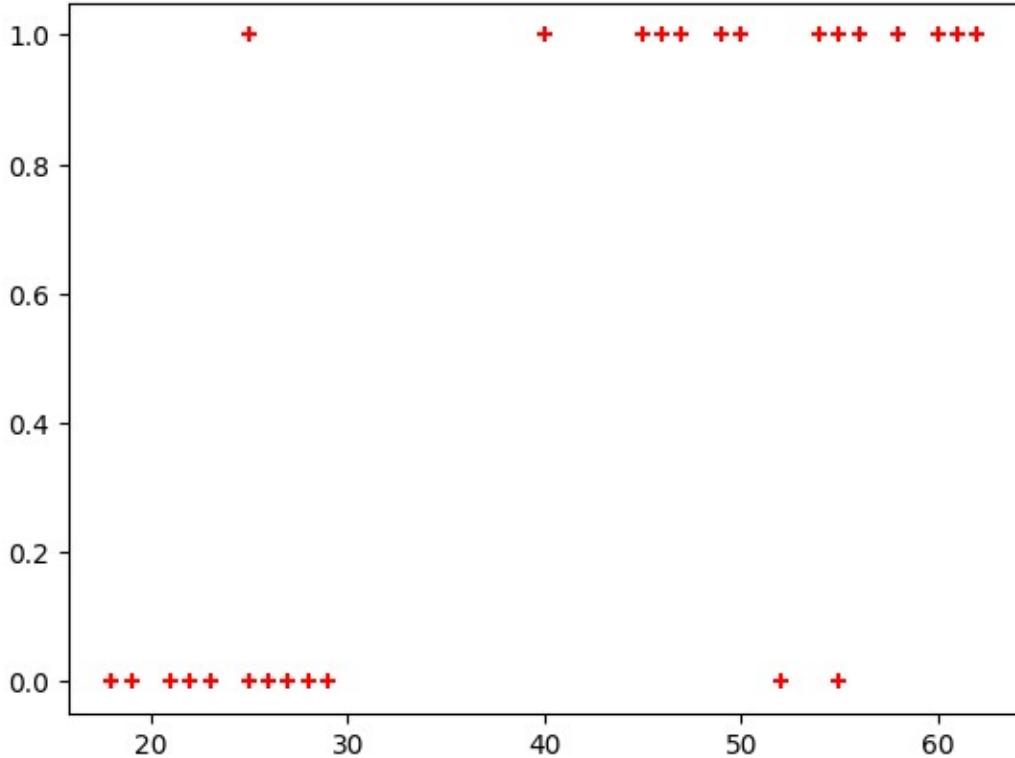
z = 0.127 * age - 4.973 # 0.12740563 ~ 0.0127 and -4.97335111 ~ -4.97
y = sigmoid(z)
return y

age = 35
prediction_function(age)

```

"""0.37 is less than 0.5 which means person with 35 will not buy the insurance"""

'0.37 is less than 0.5 which means person with 35 will not buy the insurance'



```

# Import necessary libraries
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import metrics
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = pd.read_csv("iris.csv")
iris.head()

X=iris.drop('species',axis='columns')# Features (sepal length, sepal width, petal length, petal width)
y = iris.species # Target labels (0: Setosa, 1: Versicolor, 2: Virginica)

```

```

# Split the dataset into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Multinomial Logistic Regression model
# Use 'multinomial' for multi-class classification and 'lbfgs' solver
model = LogisticRegression(multi_class='multinomial')

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate the accuracy of the model on the test data
accuracy = accuracy_score(y_test, y_pred)

# Display the accuracy
print(f"Accuracy of the Multinomial Logistic Regression model on the test set: {accuracy:.2f}")

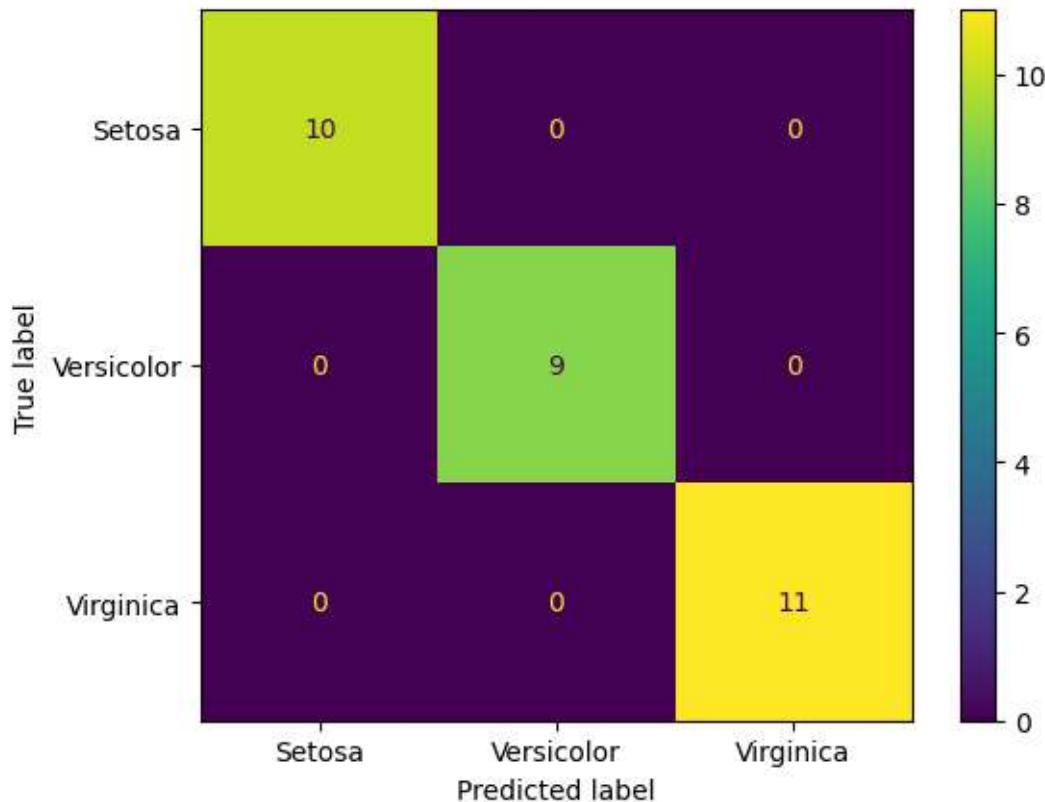
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=
= ["Setosa", "Versicolor", "Virginica"])

cm_display.plot()
plt.show()

Accuracy of the Multinomial Logistic Regression model on the test set: 1.00
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning:
'multi_class' was deprecated in version 1.5 and will be removed in 1.7. From then on, it will always
use 'multinomial'. Leave it to its default value to avoid this warning.
warnings.warn(

```



In []:

```
# Importing necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# Load the dataset
df = pd.read_csv('HR_comma_sep.csv')

# 1. Exploratory Data Analysis (EDA)

# Check the first few rows of the dataset
print(df.head())

# Check for missing values
print(df.isnull().sum())

# Describe the dataset (summary statistics)
print(df.describe())

# Check the distribution of 'left' column (Employee Retention: 1 means left, 0 means stayed)
```

```

sns.countplot(x='left', data=df)
plt.title("Employee Retention (Stayed vs Left)")
plt.show()

# 2. Impact of Employee Salaries on Retention
# Plotting bar chart showing impact of employee salaries on retention
sns.countplot(x='salary', hue='left', data=df)
plt.title("Impact of Employee Salaries on Retention")
plt.show()

# 3. Correlation between Department and Employee Retention
# Plotting bar chart showing correlation between department and retention
sns.countplot(x='Department', hue='left', data=df)
plt.title("Correlation Between Department and Employee Retention")
plt.xticks(rotation=45) # Rotate x-axis labels for better readability
plt.show()

# 4. Build Logistic Regression Model

# Convert categorical columns to numeric using one-hot encoding
df_encoded = pd.get_dummies(df, drop_first=True)

# Split the dataset into features (X) and target (y)
X = df_encoded.drop('left', axis=1) # Features (drop the 'left' column as it's the target)
y = df_encoded['left'] # Target: whether the employee left (1) or stayed (0)

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Logistic Regression model
logreg_model = LogisticRegression()

# Fit the model to the training data
logreg_model.fit(X_train, y_train)

# 5. Measure Model Accuracy

# Predict on the test set
y_pred = logreg_model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Logistic Regression Model: {accuracy * 100:.2f}%")

# Plot confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)

```

```

plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

```

	satisfaction_level	last_evaluation	number_project	average_montly_hours	\
0	0.38	0.53	2	157	
1	0.80	0.86	5	262	
2	0.11	0.88	7	272	
3	0.72	0.87	5	223	
4	0.37	0.52	2	159	

	time_spend_company	Work_accident	left	promotion_last_5years	Department	\
0	3	0	1	0	sales	
1	6	0	1	0	sales	
2	4	0	1	0	sales	
3	5	0	1	0	sales	
4	3	0	1	0	sales	

	salary
0	low
1	medium
2	medium
3	low
4	low

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	Department	salary
	0	0	0	0	0	0	0	0	0	0

satisfaction_level 0
 last_evaluation 0
 number_project 0
 average_montly_hours 0
 time_spend_company 0
 Work_accident 0
 left 0
 promotion_last_5years 0
 Department 0
 salary 0

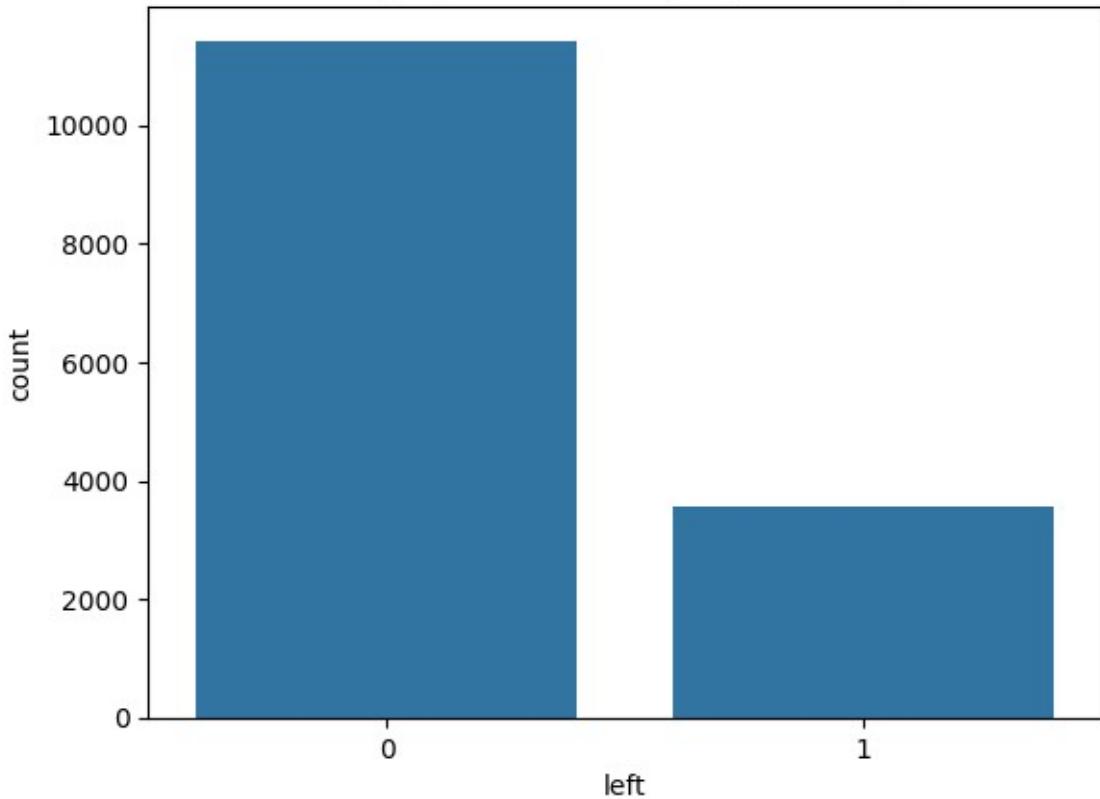
dtype: int64

	satisfaction_level	last_evaluation	number_project	\
count	14999.000000	14999.000000	14999.000000	
mean	0.612834	0.716102	3.803054	
std	0.248631	0.171169	1.232592	
min	0.090000	0.360000	2.000000	
25%	0.440000	0.560000	3.000000	
50%	0.640000	0.720000	4.000000	
75%	0.820000	0.870000	5.000000	
max	1.000000	1.000000	7.000000	

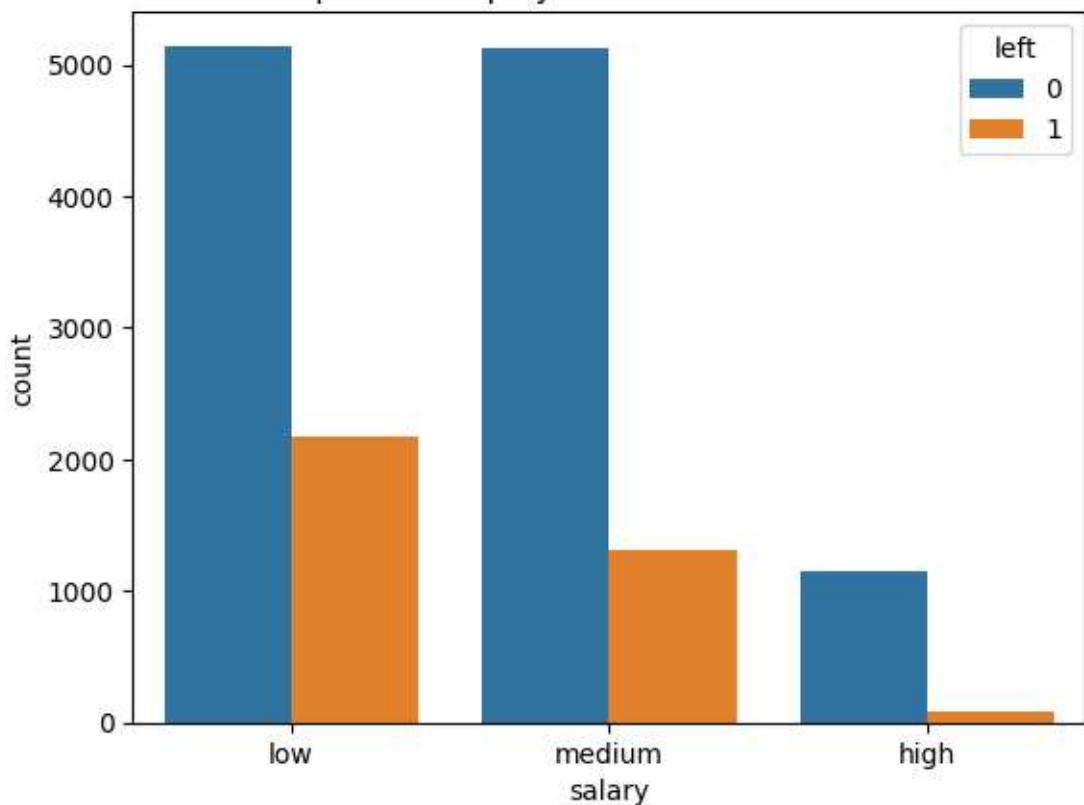
	average_montly_hours	time_spend_company	Work_accident	left \
count	14999.000000	14999.000000	14999.000000	14999.000000
mean	201.050337	3.498233	0.144610	0.238083
std	49.943099	1.460136	0.351719	0.425924
min	96.000000	2.000000	0.000000	0.000000
25%	156.000000	3.000000	0.000000	0.000000
50%	200.000000	3.000000	0.000000	0.000000
75%	245.000000	4.000000	0.000000	0.000000
max	310.000000	10.000000	1.000000	1.000000

	promotion_last_5years
count	14999.000000
mean	0.021268
std	0.144281
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	1.000000

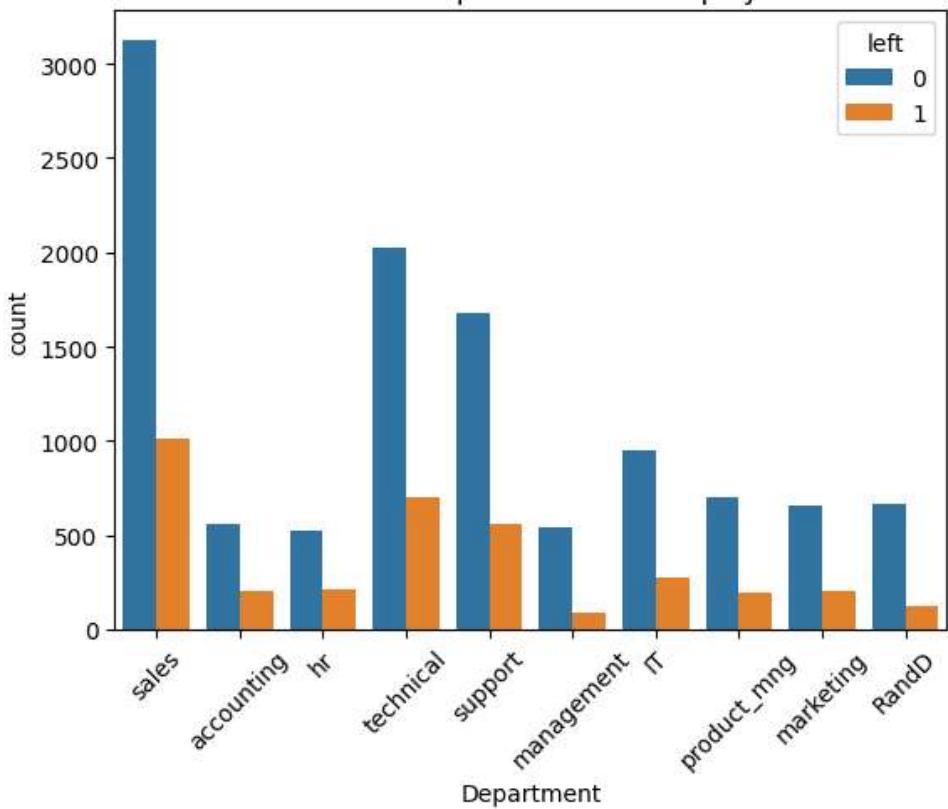
Employee Retention (Stayed vs Left)



Impact of Employee Salaries on Retention



Correlation Between Department and Employee Retention



Accuracy of Logistic Regression Model: 79.63%
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

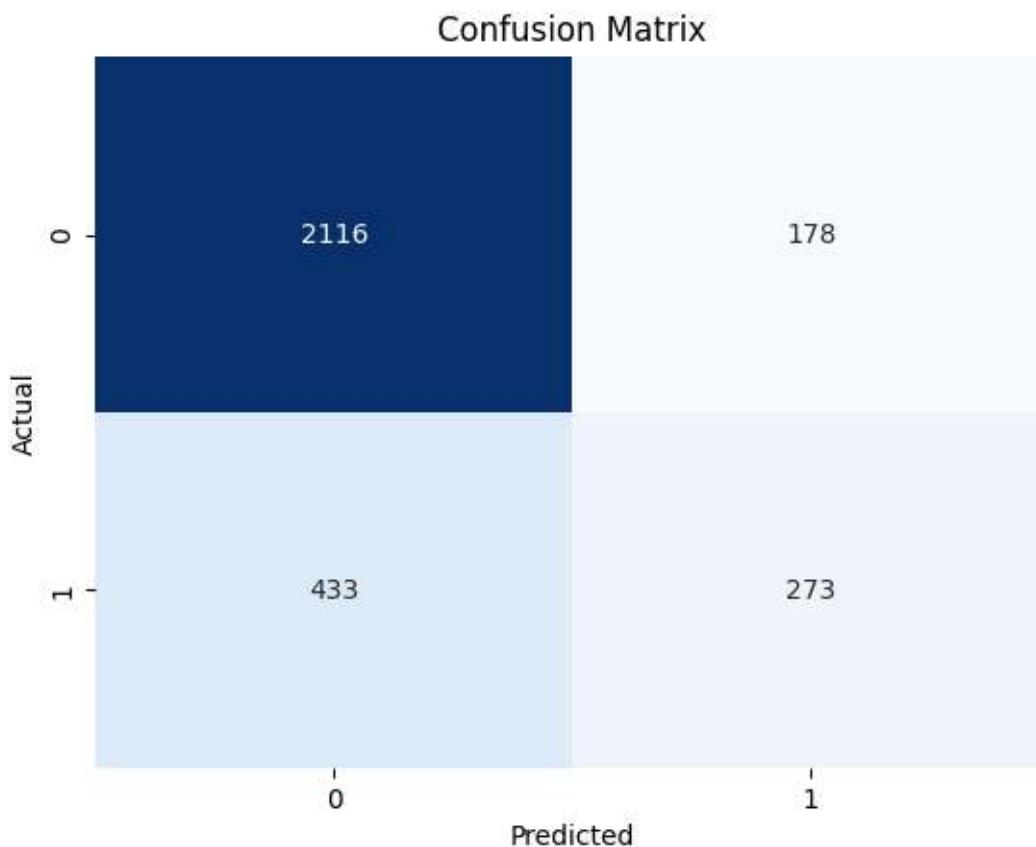
Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

n_iter_i = _check_optimize_result()



```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
import numpy as np
```

```
# Load the datasets
```

```
zoo_data = pd.read_csv('zoo-data.csv')
class_type = pd.read_csv('zoo-class-type.csv')
```

```
# 1. Data Preprocessing: Merge datasets to get class_type
```

```

zoo_data = zoo_data.merge(class_type[['Class_Number', 'Class_Type']], how='left',
left_on='class_type', right_on='Class_Number')

# Drop the unnecessary 'Class_Number' column and keep 'Class_Type' as the target variable
zoo_data.drop(columns=['Class_Number'], inplace=True)

# Check for missing values
print(zoo_data.isnull().sum())

# Since we don't have missing values, we can proceed to encode categorical variables and model building.

# 2. Feature Engineering: Encode categorical variables
# Here, we'll encode 'class_type' which is now 'Class_Type' into numeric labels using LabelEncoder
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

# Convert 'Class_Type' (target) into numerical labels
zoo_data['Class_Type'] = label_encoder.fit_transform(zoo_data['Class_Type'])

# 3. Split the data into features (X) and target (y)
X = zoo_data.drop(['animal_name', 'class_type', 'Class_Type'], axis=1) # Features
y = zoo_data['Class_Type'] # Target

# 4. Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 5. Build Logistic Regression Model
logreg_model = LogisticRegression(max_iter=1000, multi_class='ovr') # Using 'ovr' for multiclass classification
logreg_model.fit(X_train, y_train)

# 6. Predictions and Accuracy
y_pred = logreg_model.predict(X_test)

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Logistic Regression Model: {accuracy * 100:.2f}%")

# 7. Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plotting Confusion Matrix
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False,
xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.title("Confusion Matrix for Logistic Regression Model")
plt.xlabel("Predicted")

```

```
plt.ylabel("Actual")
plt.show()
```

```
animal_name    0
hair          0
feathers      0
eggs          0
milk          0
airborne      0
aquatic       0
predator      0
toothed       0
backbone      0
breathes      0
venomous      0
fins          0
legs          0
tail          0
domestic      0
catsize       0
class_type    0
Class_Type    0
dtype: int64
```

Accuracy of Logistic Regression Model: 95.24%

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:1256: FutureWarning:
'multi_class' was deprecated in version 1.5 and will be removed in 1.7. Use
OneVsRestClassifier(LogisticRegression(..)) instead. Leave it to its default value to avoid this
warning.
```

```
warnings.warn(
```

Confusion Matrix for Logistic Regression Model

	Amphibian	Bird	Bug	Fish	Invertebrate	Mammal	Reptile
Actual	2	0	0	0	0	0	
Reptile							
Amphibian -	2	0	0	0	0	0	
Bird -	0	3	0	0	0	0	
Bug -	0	0	2	0	0	0	
Fish -	0	0	0	1	0	0	
Invertebrate -	0	0	0	0	12	0	
Mammal -	0	0	1	0	0	0	

Program 6

Build KNN Classification model for a given dataset.

Algorithm

Lab - 6		KNN		
Rank	Person	Age	Salary	Tar
1	A	18	50	N
2	B	23	55	N
3	C	24	70	N
4	D	41	60	Y
5	E	43	70	Y
6	F	38	40	Y
7	X	35	100	?

$$D_{st} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$
For A, $D_{st} = \sqrt{(25 - 18)^2 + (100 - 50)^2} = 52.81$. Similarly
 For \dots

Decision Tree:

splits data based on entropy, predicts the class in each leaf node

KNN

N
114/105

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split, confusion_matrix, classification_report
from sklearn.preprocessing import LabelEncoder
data = pd.read_csv('iris.csv')
X = data.drop(['species'], axis=1)
```

y = data['species']

le = LabelEncoder()

y = le.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

accuracy = []

for k in range(1, 21):

knn = KNeighborsClassifier(n_neighbors=k)

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

accuracy.append(accuracy_score(y_test, y_pred))
optimal_k = np.argmax(accuracy) + 1

print(f'Optimal K value: {optimal_k}')

knn = KNeighborsClassifier(n_neighbors=optimal_k)

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(accuracy)

cm = confusion_matrix(y_test, y_pred)

Classification report

	Precision	recall	f-score	Sup.
Iris-setosa	1	1	1	1
Iris-versicolor	1	1	1	9
Iris-virginica	1	1	1	11
accuracy			1	30
macro avg	1	1	1	30
weighted avg	1	1	1	30

① for iris data set how to choose k?

K=3 gives 100% accuracy but generally k ≥ 5. Iris (no misclassification), to find best k, test & plot error rate

② For diabetes data what is the purpose of how do perform?

Code

IRIS DATASET

```
import pandas as pd
```

```
# Assuming the uploaded file is named 'iris.csv'  
file_path = 'iris.csv' # The path to your uploaded file
```

```
# Load the dataset into a pandas DataFrame
```

```

df = pd.read_csv(file_path)

# Display the first few rows to ensure the dataset loaded correctly
print(df.head())

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import LabelEncoder

# Assuming the dataset has columns: sepal_length, sepal_width, petal_length, petal_width, species
X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']] # Features
y = df['species'] # Target label

# Encode the target labels (species) if needed
le = LabelEncoder()
y = le.fit_transform(y)

# Split the data into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the KNN model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Make predictions on the test set
y_pred = knn.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print results
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)

```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Accuracy: 1.0

Confusion Matrix:

```
[[10 0 0]
 [ 0 9 0]
 [ 0 0 11]]
```

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	1.00	1.00	1.00	9
2	1.00	1.00	1.00	11
accuracy		1.00	1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

DIABETES PREDICTION

In [4]:

```
import pandas as pd
import numpy as np
```

In [5]:

```
data = pd.read_csv("diabetes.csv")
data.head()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [20]:

```
from sklearn.preprocessing import StandardScaler
```

```
sc_X = StandardScaler()
X = pd.DataFrame(sc_X.fit_transform(data_copy.drop(["Outcome"], axis=1)), columns=['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age'])
```

In [21]:

```
X.head()
```

Out[21]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.639947	0.865108	-0.033518	0.670643	-0.181541	0.166619	0.468492	1.425995
1	-0.844885	-1.206162	-0.529859	-0.012301	-0.181541	-0.852200	-0.365061	0.190672
2	1.233880	2.015813	-0.695306	-0.012301	-0.181541	1.332500	0.604397	0.105584
3	-0.844885	-1.074652	-0.529859	-0.695245	-0.540642	0.633881	-0.920763	1.041549
4	-1.141852	0.503458	-2.680669	0.670643	0.316566	1.549303	5.484909	-0.020496

In [22]:

```
y = data_copy.Outcome
```

In [23]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 42, stratify=y)
```

In [24]:

```
from sklearn.neighbors import KNeighborsClassifier
```

```
train_scores = []
test_scores = []
```

```

for i in range(1,15):
    knn = KNeighborsClassifier(i)
    knn.fit(X_train, y_train)
    train_scores.append(knn.score(X_train, y_train))
    test_scores.append(knn.score(X_test, y_test))

max_test_score = max(test_scores)

test_score_index = [i for i, v in enumerate(test_scores) if v == max_test_score]

print('Max test score {} % and k = {}'.format(max_test_score*100, list(map(lambda x: x+1, test_score_index))))

```

Max test score 76.5625 % and k = [11]

```

# K=11
#Setup a knn classifier with k neighbors
knn = KNeighborsClassifier(11)

knn.fit(X_train,y_train)
knn.score(X_test,y_test)

```

0.765625

In [29]:

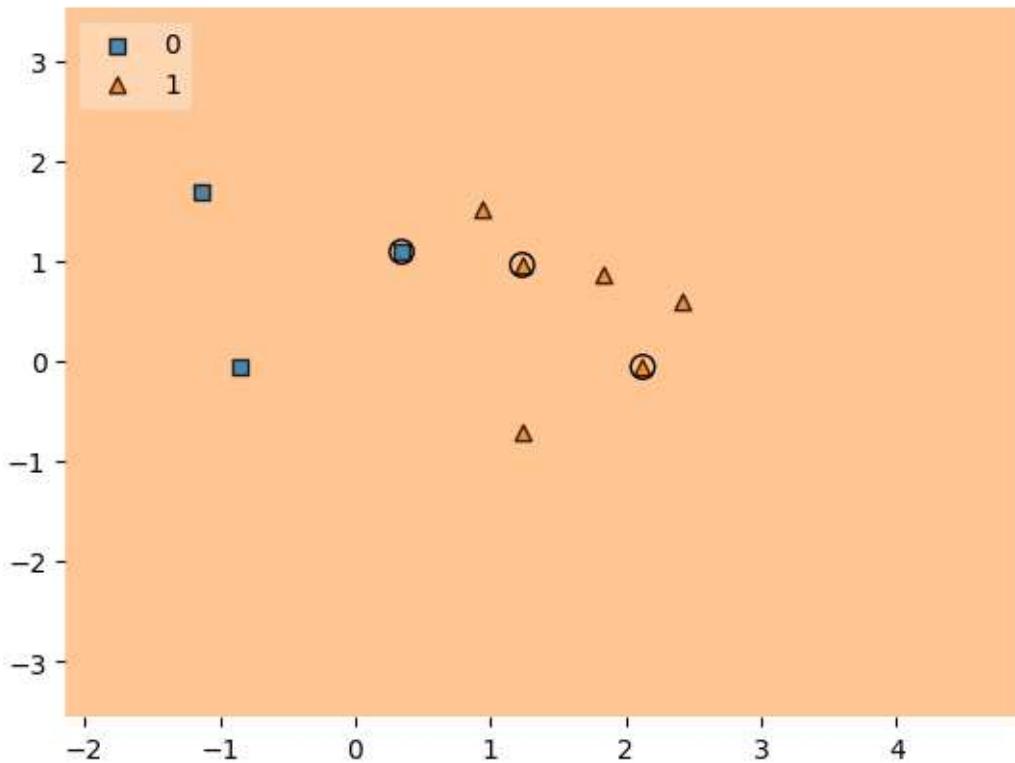
```

from mlxtend.plotting import plot_decision_regions
value = 20000
width = 20000

plot_decision_regions(X.values, y.values, clf = knn, legend = 2, filler_feature_values={2: value, 3: value, 4: value, 5: value, 6: value, 7: value},
                      filler_feature_ranges={2: width, 3: width, 4: width, 5: width, 6: width, 7: width},
                      X_highlight=X_test.values)
plt.title("KNN with diabetes data")
plt.show()

```

KNN with diabetes data

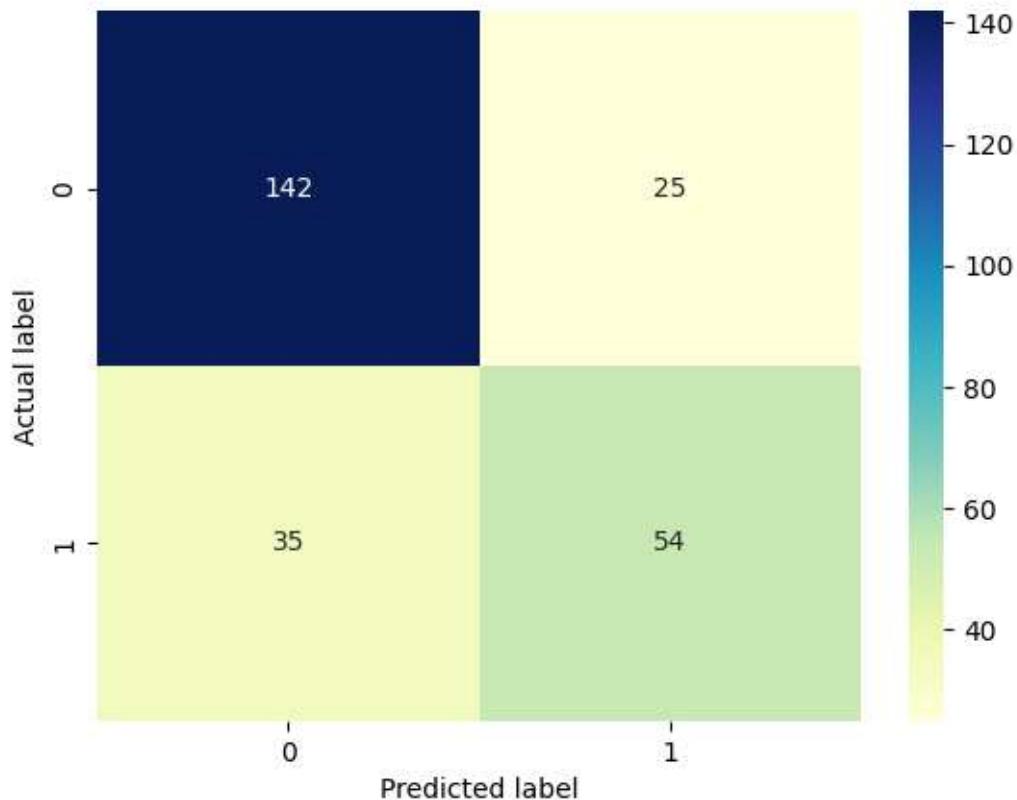


```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, fbeta_score
y_pred = knn.predict(X_test)

cnf_matrix = confusion_matrix(y_test, y_pred)

p = sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
Text(0.5, 23.52222222222222, 'Predicted label')
```

Confusion matrix



```
def model_evaluation(y_test, y_pred, model_name):
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    f2 = fbeta_score(y_test, y_pred, beta = 2.0)

    results = pd.DataFrame([[model_name, acc, prec, rec, f1, f2]],
                           columns = ["Model", "Accuracy", "Precision", "Recall",
                                      "F1 Score", "F2 Score"])
    results = results.sort_values(["Precision", "Recall", "F2 Score"], ascending = False)
    return results
```

```
model_evaluation(y_test, y_pred, "KNN")
```

Out[32]:

Model	Accuracy	Precision	Recall	F1 Score	F2 Score
-------	----------	-----------	--------	----------	----------

Model	Accuracy	Precision	Recall	F1 Score	F2 Score
0 KNN	0.765625	0.683544	0.606742	0.642857	0.62069

HEART DISEASE PREDICTION

```

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler

# Load the dataset
df = pd.read_csv('heart.csv')

# Define features (X) and target (y)
X = df.drop('target', axis=1)
y = df['target']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Find the best k value
best_k = 1
best_accuracy = 0
for k in range(1, 31): # Test k values from 1 to 30
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_k = k

print(f"Best k: {best_k}")

# Train the KNN classifier with the best k value
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)

# Make predictions

```

```

y_pred = knn.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Display results
print(f'Accuracy: {accuracy}')
print(f'Confusion Matrix:\n{conf_matrix}')
print(f'Classification Report:\n{class_report}')

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Disease', 'Disease'],
            yticklabels=['No Disease', 'Disease'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Plot classification report (text-based, can't be plotted directly)
print("Classification Report:\n", class_report)

```

Best k: 7

Accuracy: 0.9180327868852459

Confusion Matrix:

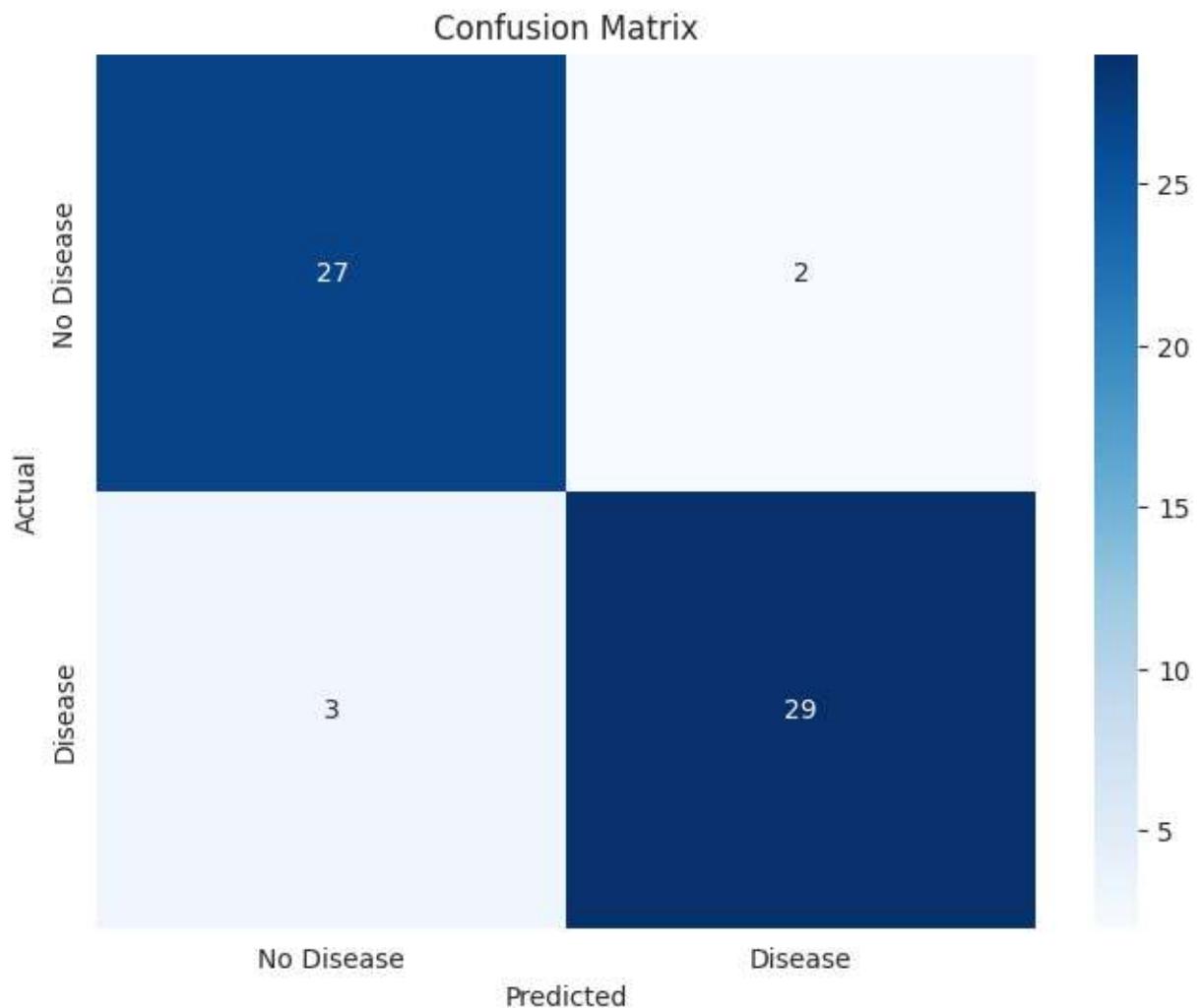
```

[[27  2]
 [ 3 29]]

```

Classification Report:

	precision	recall	f1-score	support
0	0.90	0.93	0.92	29
1	0.94	0.91	0.92	32
accuracy			0.92	61
macro avg	0.92	0.92	0.92	61
weighted avg	0.92	0.92	0.92	61



Classification Report:

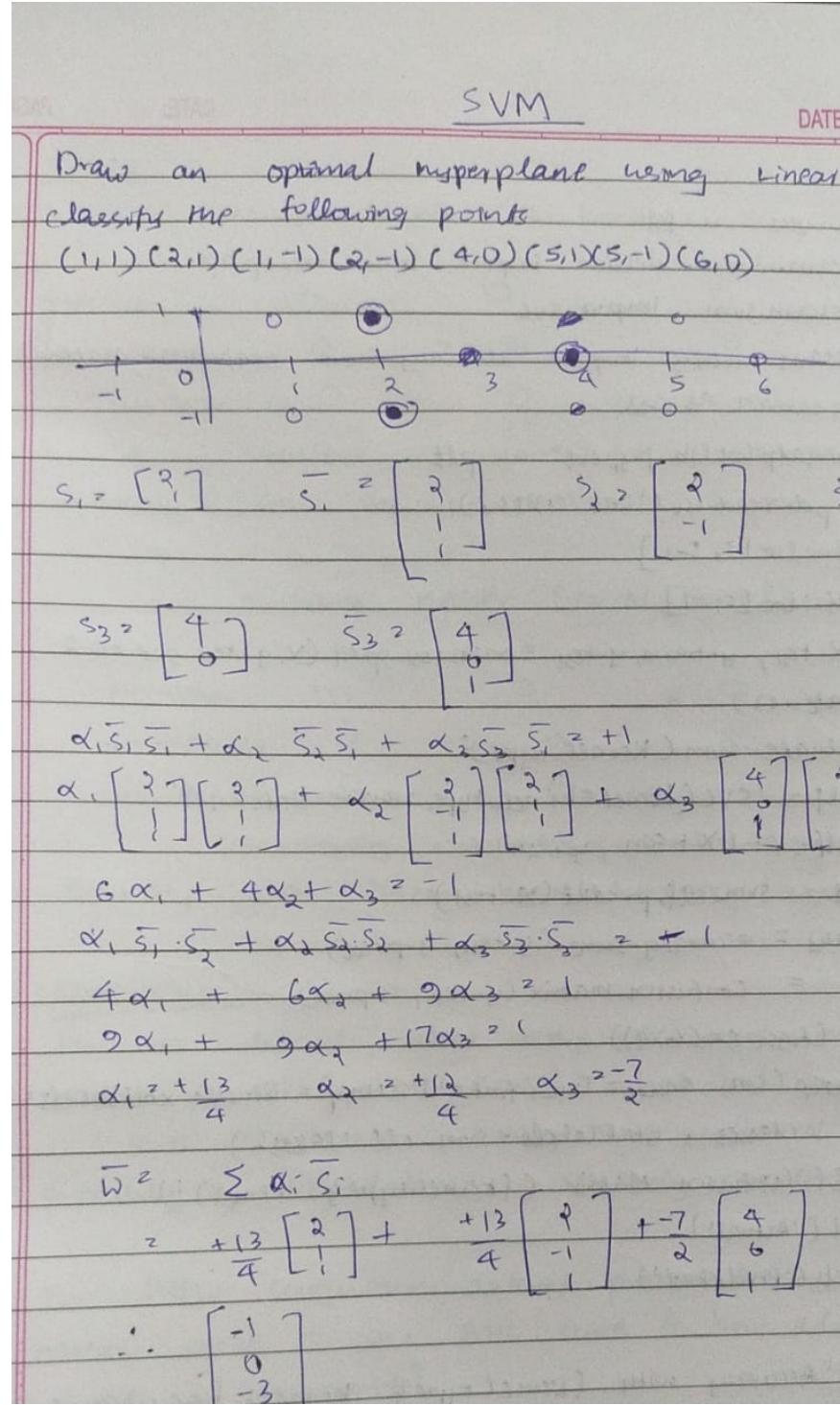
	precision	recall	f1-score	support
0	0.90	0.93	0.92	29
1	0.94	0.91	0.92	32
accuracy		0.92	0.92	61
macro avg	0.92	0.92	0.92	61
weighted avg	0.92	0.92	0.92	61

In []:

Program 7

Build Support vector machine model for a given dataset

Algorithm



```

This
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion
import seaborn as sns
import matplotlib.pyplot as plt
iris_df = pd.read_csv('iris.csv')
X = iris_df.iloc[:, :-1]
y = iris_df.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
def evaluate_svm(kernel_type):
    svm_clf = SVC(kernel=kernel_type, random_state=42)
    svm_clf.fit(X_train, y_train)
    y_pred = svm_clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                classes, yticklabels=svm_clf.classes_)
    plt.title(f'Confusion Matrix ({kernel_type} kernel)')
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()
    print(f"Accuracy with {kernel_type}: {accuracy}")

```

Output:

Confusion Matrix (rbf kernel)

Actual	iris-setosa	10	0	0
Predicted	iris-versicolor	0	9	0
	iris-virginica	0	0	11
		iris-setosa	iris-versicolor	iris-virginica
Accuracy	with rbf kernel	1.0000		

Confusion Matrix (linear kernel)

iris-setosa	10	0	0
iris-versicolor	0	9	0
iris-virginica	0	0	11
	iris-setosa	iris-versicolor	iris-virginica
Accuracy	with linear kernel	1.0000	

Questions

- For iris dataset what is the accuracy score using linear & RBF kernel? Which has better
Both has 1.0000 accuracy. Both performed eq
probably as the dataset is relatively small &
- For letter recognition dataset present & interpret
matrix what is the AUC score & how does it
model performance?

AUC: 0.9081

Code:

```
# Import necessary libraries
import pandas as pd
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# URL of the Iris dataset (adjust this URL if needed)
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# Column names as the dataset contains headers now
columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']

# Load the dataset from the URL into a pandas DataFrame
iris = pd.read_csv(url, header=None, names=columns)

# Map the species names to numeric values (for classification)
iris['species'] = iris['species'].map({'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2})

# Split features and target
X = iris.drop('species', axis=1)
y = iris['species']

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train SVM with a linear kernel
svm_linear = SVC(kernel='linear', random_state=42)
svm_linear.fit(X_train, y_train)

# Predict and calculate accuracy for the linear kernel model
y_pred_linear = svm_linear.predict(X_test)
accuracy_linear = accuracy_score(y_test, y_pred_linear)
conf_matrix_linear = confusion_matrix(y_test, y_pred_linear)
```

```

# Initialize and train SVM with RBF kernel
svm_rbf = SVC(kernel='rbf', random_state=42)
svm_rbf.fit(X_train, y_train)

# Predict and calculate accuracy for the RBF kernel model
y_pred_rbf = svm_rbf.predict(X_test)
accuracy_rbf = accuracy_score(y_test, y_pred_rbf)
conf_matrix_rbf = confusion_matrix(y_test, y_pred_rbf)

# Display accuracy scores
print(f"Accuracy of SVM with Linear Kernel: {accuracy_linear:.2f}")
print(f"Accuracy of SVM with RBF Kernel: {accuracy_rbf:.2f}")

# Plot confusion matrices for both models
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Linear Kernel Confusion Matrix
sns.heatmap(conf_matrix_linear, annot=True, fmt="d", cmap="Blues",
            xticklabels=iris['species'].unique(),
            yticklabels=iris['species'].unique(), ax=axes[0])
axes[0].set_title('Confusion Matrix - Linear Kernel')

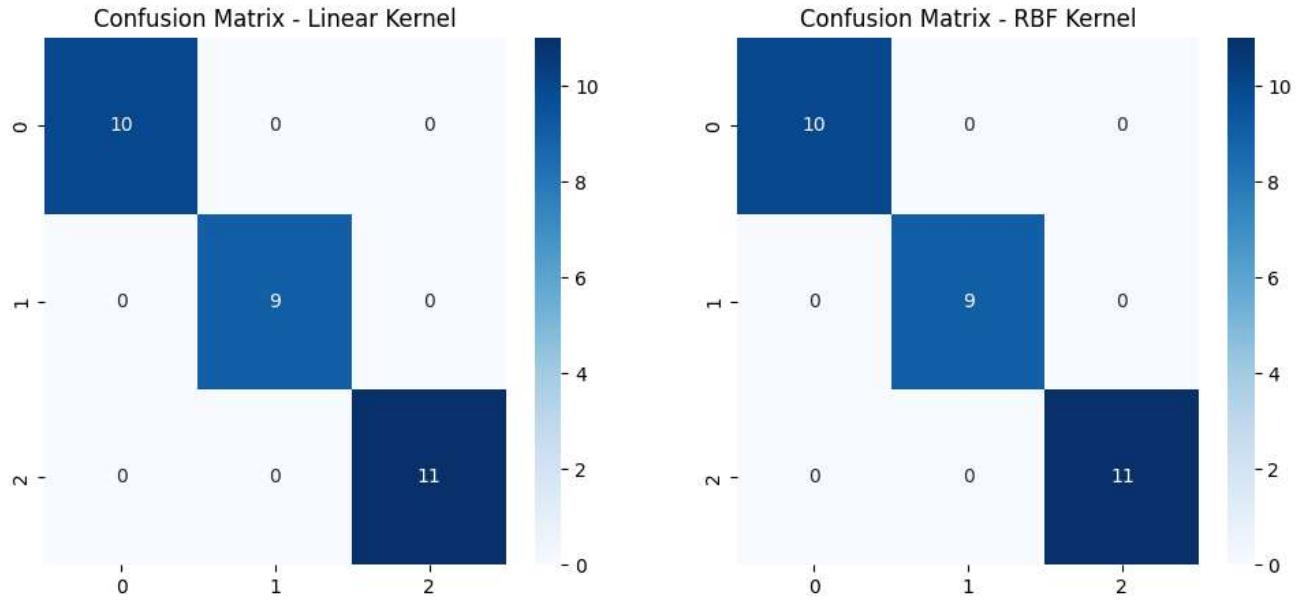
# RBF Kernel Confusion Matrix
sns.heatmap(conf_matrix_rbf, annot=True, fmt="d", cmap="Blues",
            xticklabels=iris['species'].unique(),
            yticklabels=iris['species'].unique(), ax=axes[1])
axes[1].set_title('Confusion Matrix - RBF Kernel')

plt.show()

```

Accuracy of SVM with Linear Kernel: 1.00

Accuracy of SVM with RBF Kernel: 1.00



```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, auc
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.preprocessing import label_binarize

# Load the Letter-recognition dataset
letter_data = pd.read_csv('letter-recognition.csv')

# Prepare the dataset
X_letter = letter_data.drop('letter', axis=1)
y_letter = letter_data['letter']

# Encode the letter labels to numerical values
le = LabelEncoder()
y_letter_encoded = le.fit_transform(y_letter)

# Split the data into training and testing sets (80% training, 20% testing)
X_train_letter, X_test_letter, y_train_letter, y_test_letter = train_test_split(X_letter,
y_letter_encoded, test_size=0.2, random_state=42)

# Train the SVM model with a linear kernel

```

```

svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train_letter, y_train_letter)

# Make predictions
y_pred_letter = svm_linear.predict(X_test_letter)

# Calculate accuracy score
accuracy = accuracy_score(y_test_letter, y_pred_letter)
print(f'Accuracy Score with Linear Kernel: {accuracy:.4f}')

# Confusion Matrix
conf_matrix = confusion_matrix(y_test_letter, y_pred_letter)

# Plot Confusion Matrix (without printing the elements)
plt.figure(figsize=(10, 8)) # Make the plot larger
ConfusionMatrixDisplay(conf_matrix, display_labels=le.classes_).plot(cmap='Blues')
plt.title('Confusion Matrix for Letter-recognition Dataset')
plt.show()

# Binarize the labels for multi-class classification
y_test_letter_binarized = label_binarize(y_test_letter, classes=np.arange(26)) # 26 classes
y_pred_prob_letter = svm_linear.decision_function(X_test_letter)

# Compute ROC curve and AUC score for each class
fpr = {}
tpr = {}
roc_auc = {}
for i in range(26): # 26 classes
    fpr[i], tpr[i], _ = roc_curve(y_test_letter_binarized[:, i], y_pred_prob_letter[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve for each class
plt.figure(figsize=(10, 8))
for i in range(26):
    plt.plot(fpr[i], tpr[i], lw=2, label=f'Class {i} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for Letter-recognition Dataset (Multi-class)')
plt.legend(loc="lower right")

```

```

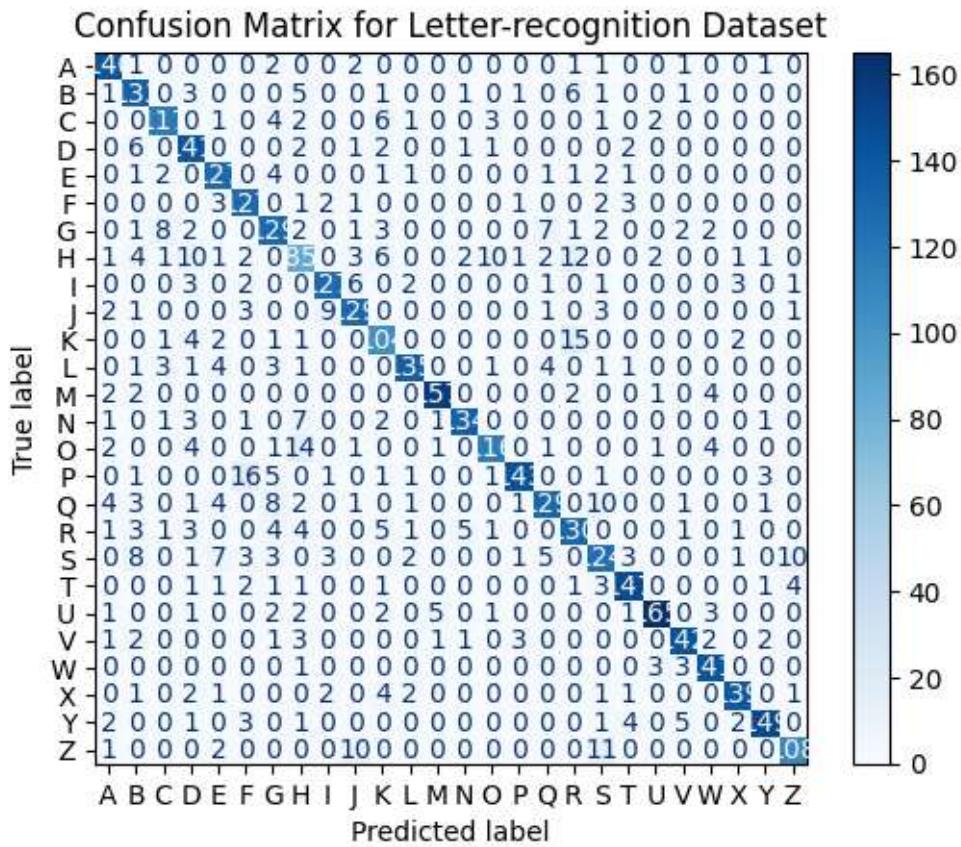
plt.show()

# Display AUC scores for each class
for i in range(26):
    print(f'AUC score for Class {i}: {roc_auc[i]:.4f}')

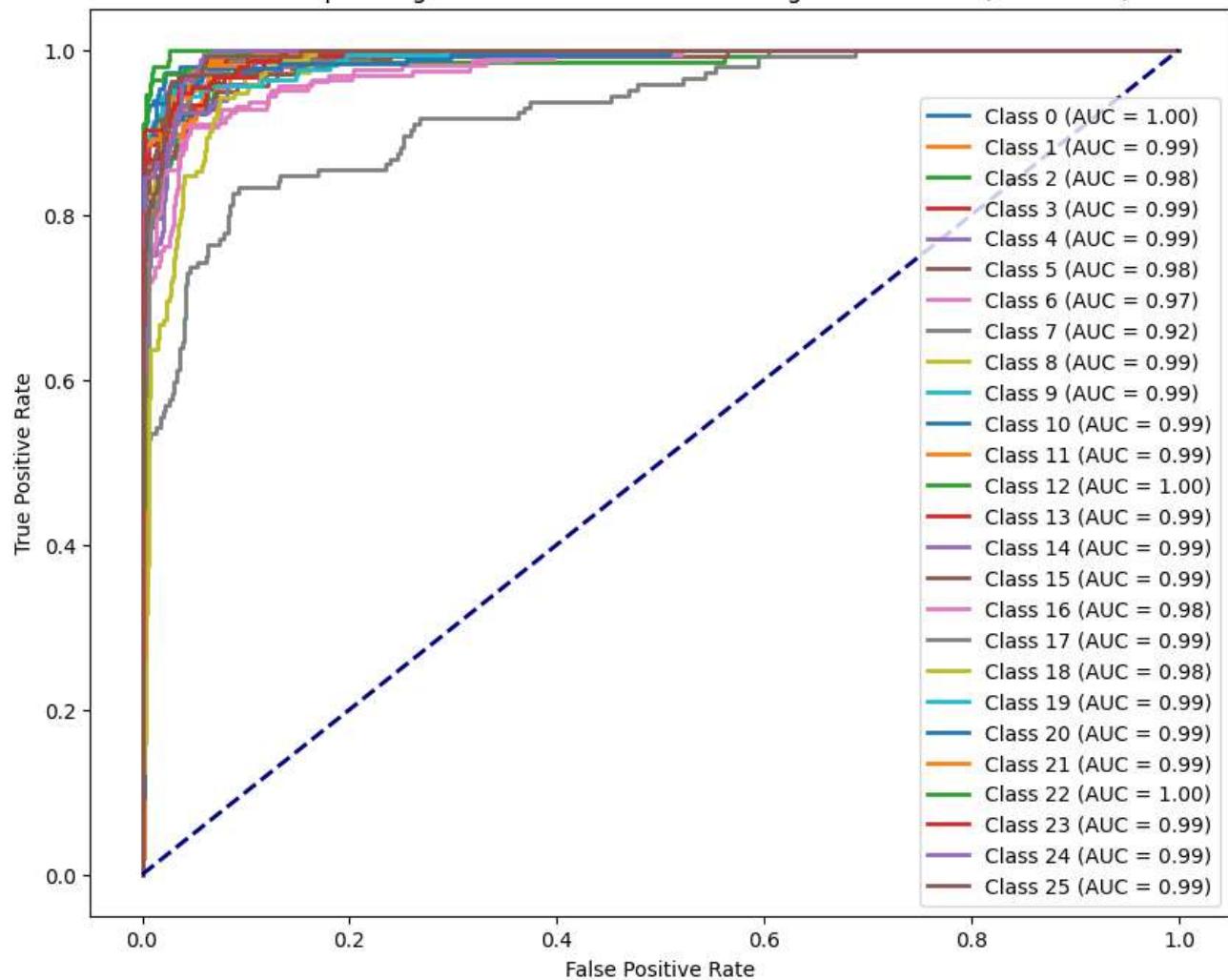
```

Accuracy Score with Linear Kernel: 0.8545

<Figure size 1000x800 with 0 Axes>



Receiver Operating Characteristic for Letter-recognition Dataset (Multi-class)



AUC score for Class 0: 0.9955

AUC score for Class 1: 0.9896

AUC score for Class 2: 0.9825

AUC score for Class 3: 0.9938

AUC score for Class 4: 0.9915

AUC score for Class 5: 0.9825

AUC score for Class 6: 0.9742

AUC score for Class 7: 0.9238

AUC score for Class 8: 0.9888

AUC score for Class 9: 0.9925

AUC score for Class 10: 0.9864

AUC score for Class 11: 0.9908

AUC score for Class 12: 0.9973

AUC score for Class 13: 0.9918

AUC score for Class 14: 0.9855

AUC score for Class 15: 0.9909

```
AUC score for Class 16: 0.9766
AUC score for Class 17: 0.9882
AUC score for Class 18: 0.9779
AUC score for Class 19: 0.9893
AUC score for Class 20: 0.9906
AUC score for Class 21: 0.9934
AUC score for Class 22: 0.9979
AUC score for Class 23: 0.9931
AUC score for Class 24: 0.9936
AUC score for Class 25: 0.9934
```

```
In [13]:
```

```
import pandas as pd
```

```
# Data
```

```
data = {
    "Height": [44, 52.1, 57.1, 33, 27.8, 27.2, 32, 45.1, 56.7, 56.9, 122.1],
    "Weight": [126.3, 136.9, 109.2, 148.3, 110.4, 107.8, 128.4, 120.2, 140.2, 139.2, 154.1]
}
```

```
# Create a DataFrame
```

```
df = pd.DataFrame(data)
```

```
# Save to CSV
```

```
df.to_csv('height_weight_data.csv', index=False)
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
# Load the data from CSV
```

```
df = pd.read_csv('height_weight_data.csv')
```

```
# Plotting the data
```

```
plt.scatter(df['Height'], df['Weight'], color='blue', label='Data points')
```

```
plt.xlabel('Height')
```

```
plt.ylabel('Weight')
```

```
plt.title('Height vs Weight')
```

```
plt.grid(True)
```

```
plt.show()
```

```
from sklearn.svm import SVR
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import mean_squared_error
```

```

# Define features and target
X = df[['Height']] # Feature: Height
y = df['Weight'] # Target: Weight

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create the model
svr = SVR(kernel='linear')

# Fit the model
svr.fit(X_train, y_train)

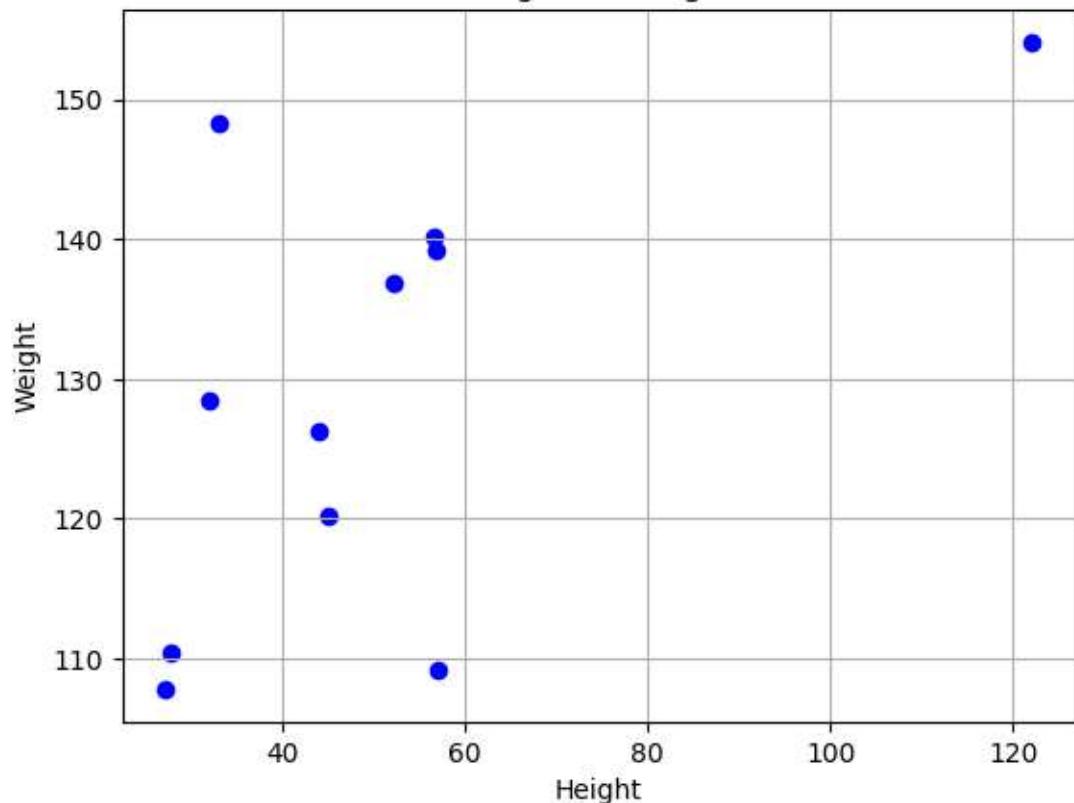
# Predict on the test set
y_pred = svr.predict(X_test)

# Calculate accuracy using Mean Squared Error (since it's regression)
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5 # Root Mean Squared Error

# Print the support vectors and accuracy
print(f"Support Vectors: {svr.support_}")
print(f"Root Mean Squared Error: {rmse}")

```

Height vs Weight



Support Vectors: [0 1 2 3 4 5 6 7]

Root Mean Squared Error: 11.680171349283054

Program 8

Implement Random forest ensemble method on a given dataset.

Algorithm

Lab-8

Random Forest Classifier

DATE:

1)

Decision Tree classifier

Definition: * A single tree based model that splits data into branches to make decisions.

Structure: * One tree

Accuracy: * Prone to overfitting, generally less accurate

Training time: * Faster since only one tree is built

Interpretability: * Easier to interpret and visualize

Random Forest classifier

An ensemble of multiple trees working together to predict accuracy.

Forest Collection

More accurate due to multiple trees, reduces error. Slower as multiple trees built and trained.

Difficult to visualize multiple trees

2) Parameters of Random Forest:

* n_estimators: Number of trees in the forest.

* max_depth: Maximum depth of each tree.

* min_samples_split: Minimum number of samples to split an internal node.

* max_features: Number of features to consider for the best split.

* bootstrap: Whether bootstrap samples are used for building trees.

* criterion: Function to measure the quality of splits.
Eg: "Gini", "entropy"

* random_state: Controls randomness

3) Algorithm

DATE:

- ⑥ Repeat recursively until max-depth is reached
⑦ Repeat steps ③ to ⑤ to build T trees

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
df = pd.read_csv('train.csv')
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
for column in X.columns:
 if X[column].dtype == 'object':
 le = preprocessing.LabelEncoder()
 X[column] = le.fit_transform(X[column])

if y.dtype == 'object':
 le = preprocessing.LabelEncoder()
 y = le.fit_transform(y)

X-train, X-test, y-train, y-test = train_test_split(X, y,
random_state=42)
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X-train, y-train)
y-pred = rf_classifier.predict(X-test)
accuracy = accuracy_score(y-test, y-pred)

DATE:

Confusion Matrix

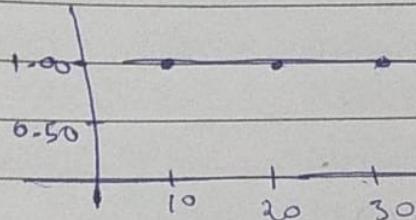
$$\begin{bmatrix} 26 & 0 & 17 \\ 0 & 12 & 5 \\ 0 & 0 & 117 \end{bmatrix}$$

AUC Score when estimators are

$$10 : 1.00$$

$$20 : 1.00$$

$$30 : 1.00$$



		Predicted	
		26	17
Actual	26	26	17
	283	2	134

$$\text{Accuracy} = \frac{26+134}{134+26+17+2} = \frac{16}{17}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{26}{26+2}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{26}{26+17}$$

$$\text{F1 score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

NP
NLP
NLTK

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
```

```

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv('train.csv')

# Preprocess the dataset
# Convert categorical columns into numeric
# For simplicity, we'll drop columns that are not necessary for modeling (e.g., 'Name', 'Ticket', 'Cabin')
df = df.drop(columns=['Name', 'Ticket', 'Cabin'])

# Convert categorical features into numeric values (Sex, Embarked)
label_encoder = LabelEncoder()
df['Sex'] = label_encoder.fit_transform(df['Sex'])
df['Embarked'] = df['Embarked'].map({'S': 0, 'C': 1, 'Q': 2}) # 'S', 'C', 'Q' to numeric

# Handle missing values (impute with the median for simplicity)
df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True) # Mode is used for Embarked

# Features and target variable
X = df.drop(columns=['Survived', 'PassengerId'])
y = df['Survived']

# Split the data into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_classifier.fit(X_train, y_train)

# Predict on the test set
y_pred = rf_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')

# Display confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)

```

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
df['Age'].fillna(df['Age'].median(), inplace=True)
<ipython-input-11-ca7102c51cf1>:22: FutureWarning: A value is trying to be set on a copy of a
DataFrame or Series through chained assignment using an inplace method.
```

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True) # Mode is used for Embarked
```

Accuracy: 0.8212

Confusion Matrix:

```
[[91 14]
 [18 56]]
```

In [12]:

```
# Import necessary libraries
```

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

```
# Load the Iris dataset
```

```
iris = load_iris()
X = iris.data
y = iris.target
```

```
# Split the dataset into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Initialize RandomForestClassifier with default n_estimators=10
clf = RandomForestClassifier(n_estimators=10, random_state=42)
```

```
# Train the model
```

```
clf.fit(X_train, y_train)
```

```
# Predict on the test set
```

```

y_pred = clf.predict(X_test)

# Evaluate the performance
score_default = accuracy_score(y_test, y_pred)
print(f"Accuracy with n_estimators=10: {score_default}")

# Fine-tune the model by changing the value of n_estimators
n_estimators_values = [10, 50, 100, 200, 500]
scores = []

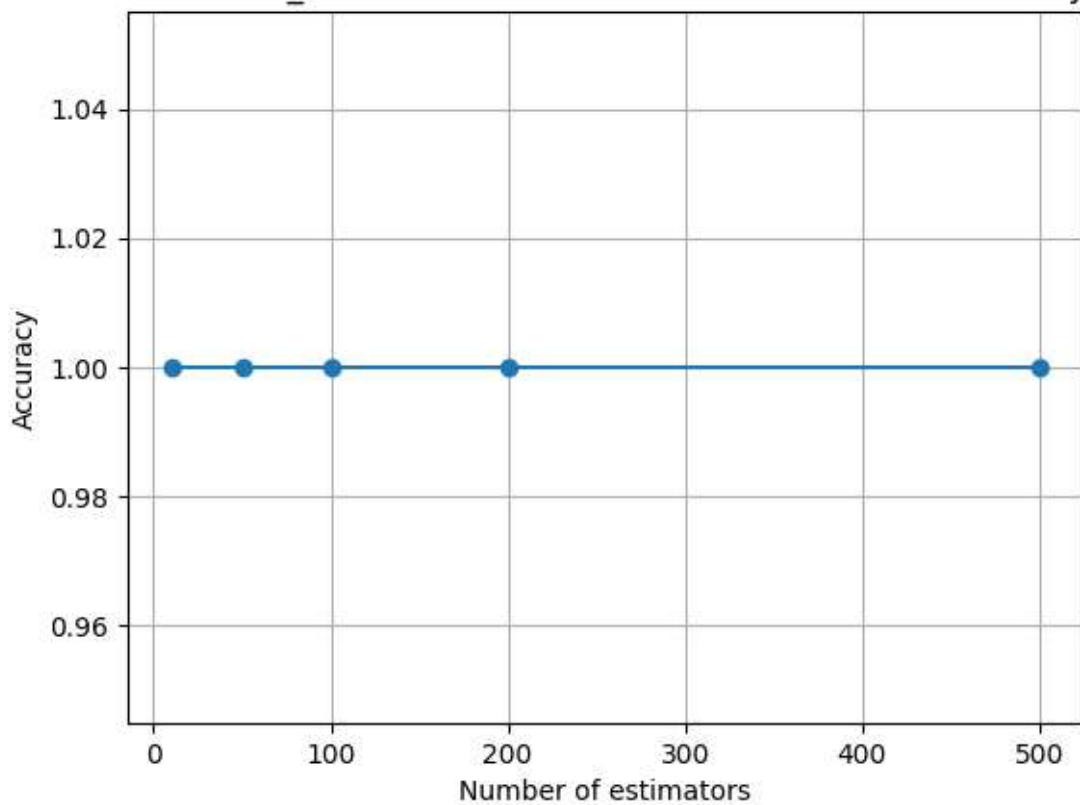
for n in n_estimators_values:
    clf = RandomForestClassifier(n_estimators=n, random_state=42)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    scores.append(score)
    print(f"Accuracy with n_estimators={n}: {score}")

# Plot the results
plt.plot(n_estimators_values, scores, marker='o')
plt.xlabel('Number of estimators')
plt.ylabel('Accuracy')
plt.title('Effect of n_estimators on Random Forest Classifier Accuracy')
plt.grid(True)
plt.show()

```

Accuracy with n_estimators=10: 1.0
 Accuracy with n_estimators=10: 1.0
 Accuracy with n_estimators=50: 1.0
 Accuracy with n_estimators=100: 1.0
 Accuracy with n_estimators=200: 1.0
 Accuracy with n_estimators=500: 1.0

Effect of n_estimators on Random Forest Classifier Accuracy



In []:

Program 9

Implement Boosting ensemble method on a given dataset.

Algorithm

What is boosting?

Boosting combines multiple weak learners to create a strong learner. It works by training models sequentially where each model corrects the errors made by previous ones.

Parameters

estimator - The base model

n_estimators - No. of weak learners

learning_rate - Shinton's contribution of each learner

algorithm -

random_state - for reproducibility.

Algorithm

1. Start with equal weights for all training samples
2. Train a weak model
3. Calculate error & update sample weights
4. Add weak model to ensemble which will be used for prediction
5. Repeat for n-estimators
6. Final prediction

Code : import pandas as pd

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.ensemble import AdaBoostClassifier
```

```
from sklearn.metrics import accuracy_score, confusion_matrix
```

```
import data = pd.read_csv('income.csv')
```

```
x = data.drop('income-level', axis=1)
```

```

print(f"Accuracy : {accuracy}%")
cm = confusion_matrix(y-test, y-pred)
print(cm)

```

Output: Accuracy: 0.8327
 [[7003, 411],
 [1223, 1132]]

iris dataset

accuracy with n-estimators:

10 : 1.00

50 : 1.00

100 : 1.00

200 : 1.00

500 : 1.00

Best n-estimator : 10

Best accuracy score: 1.00

For both ~~linear~~^{logistic} regression and decision tree,
 n-estimator was 10 at learning rates of
 respectively.

Code

import pandas as pd

```

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
# Load the data (assume "income.csv" is already downloaded)
df = pd.read_csv('income.csv')

# Let's check the columns to see what we're dealing with
print(df.head())

# Preprocessing: Encode categorical data if necessary (in this case 'income_level' is categorical)
encoder = LabelEncoder()
df['income_level'] = encoder.fit_transform(df['income_level'])

# Splitting the data into features (X) and target (y)
X = df.drop('income_level', axis=1)
y = df['income_level']

# Split the data into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

```

age fnlwgt education_num capital_gain capital_loss hours_per_week \

0	39	77516	13	2174	0	40
1	50	83311	13	0	0	13
2	38	215646	9	0	0	40
3	53	234721	7	0	0	40
4	28	338409	13	0	0	40

income_level

0	0
1	0
2	0
3	0
4	0

Initialize the AdaBoost model with Random Forest as the base classifier

```

ada_boost = AdaBoostClassifier(RandomForestClassifier(n_estimators=10, max_depth=5,
random_state=42), n_estimators=50, learning_rate=1.0)

# Train the AdaBoost model
ada_boost.fit(X_train, y_train)

# Predictions
y_pred = ada_boost.predict(X_test)

# Evaluate accuracy and confusion matrix
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")

```

Accuracy: 0.8368307912785341

Confusion Matrix:

```

[[7039 375]
 [1219 1136]]

```

```

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

```

Load the Iris dataset

```

iris = load_iris()
X = iris.data
y = iris.target

```

Split the dataset into 80% training and 20% testing

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
base_classifier = RandomForestClassifier(n_estimators=10, max_depth=5, random_state=42)
```

Initialize AdaBoost with Decision Tree as base classifier

```
ada_boost = AdaBoostClassifier(base_classifier, n_estimators=50)
```

Train the model

```

ada_boost.fit(X_train, y_train)

# Evaluate accuracy on the test set
y_pred = ada_boost.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy of AdaBoost on Iris dataset: {accuracy}')

```

Accuracy of AdaBoost on Iris dataset: 1.0

```

# Varying n_estimators and measuring the accuracy
estimators = [10, 50, 100, 200, 500]
for n in estimators:
    ada_boost = AdaBoostClassifier(base_classifier, n_estimators=n)
    ada_boost.fit(X_train, y_train)
    y_pred = ada_boost.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f'Accuracy with n_estimators={n}: {accuracy}')

```

Accuracy with n_estimators=10: 1.0

Accuracy with n_estimators=50: 1.0

Accuracy with n_estimators=100: 1.0

Accuracy with n_estimators=200: 1.0

Accuracy with n_estimators=500: 1.0

In [36]:

```

# Varying n_estimators and learning_rate
learning_rates = [0.1, 0.5, 1.0, 1.5]
n_estimators_values = [50, 100, 200]

for lr in learning_rates:
    for n in n_estimators_values:
        ada_boost = AdaBoostClassifier(base_classifier,
                                       n_estimators=n, learning_rate=lr)
        ada_boost.fit(X_train, y_train)
        y_pred = ada_boost.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        print(f'Accuracy with n_estimators={n} and learning_rate={lr}: {accuracy}')

```

Accuracy with n_estimators=50 and learning_rate=0.1: 1.0

```
Accuracy with n_estimators=100 and learning_rate=0.1: 1.0
Accuracy with n_estimators=200 and learning_rate=0.1: 1.0
Accuracy with n_estimators=50 and learning_rate=0.5: 1.0
Accuracy with n_estimators=100 and learning_rate=0.5: 1.0
Accuracy with n_estimators=200 and learning_rate=0.5: 1.0
Accuracy with n_estimators=50 and learning_rate=1.0: 1.0
Accuracy with n_estimators=100 and learning_rate=1.0: 1.0
Accuracy with n_estimators=200 and learning_rate=1.0: 1.0
Accuracy with n_estimators=50 and learning_rate=1.5: 1.0
Accuracy with n_estimators=100 and learning_rate=1.5: 1.0
Accuracy with n_estimators=200 and learning_rate=1.5: 1.0
```

In [37]:

```
from sklearn.linear_model import LogisticRegression
```

```
# Using Logistic Regression as base estimator in AdaBoost
ada_boost_lr = AdaBoostClassifier(LogisticRegression(max_iter=1000), n_estimators=50)
ada_boost_lr.fit(X_train, y_train)
y_pred_lr = ada_boost_lr.predict(X_test)
accuracy_lr = accuracy_score(y_test, y_pred_lr)
print(f'Accuracy with Logistic Regression base estimator: {accuracy_lr}')
```

Accuracy with Logistic Regression base estimator: 0.9333333333333333

In [38]:

```
# Using Decision Tree as base estimator in AdaBoost
ada_boost_tree = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1), n_estimators=50)
ada_boost_tree.fit(X_train, y_train)
y_pred_tree = ada_boost_tree.predict(X_test)
accuracy_tree = accuracy_score(y_test, y_pred_tree)
print(f'Accuracy with Decision Tree base estimator: {accuracy_tree}')
```

Accuracy with Decision Tree base estimator: 0.9333333333333333

Program 10

Build k-Means algorithm to cluster a set of data stored in a .CSV file.

Algorithm

Algorithm

1. Randomly initialize k points as centroids
2. Categorize each item to its closest mean and update mean's coordinates which are the means of the categorized in that cluster so far
3. Repeat for a given number of iterations
4. Predict the cluster by taking the closest distance centroid by σ for the point.

Code → Iris dataset

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
X = df[['petal length (cm)', 'petal width (cm)']].values
sse_scaled = []

for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    sse_scaled.append(kmeans.inertia_)

```

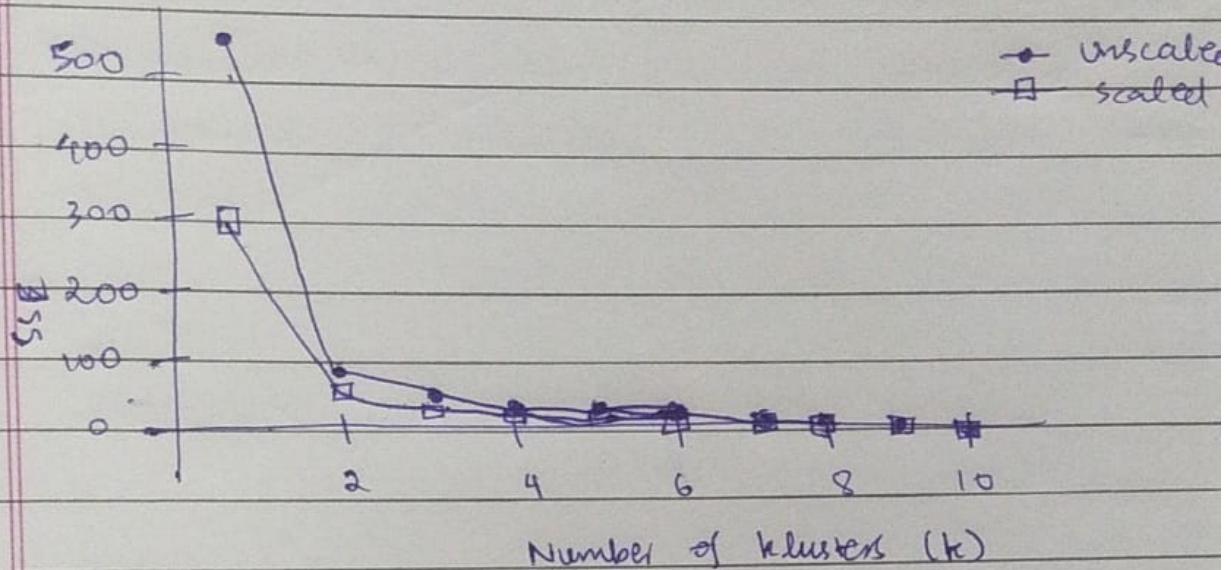
DATE:

```
plt.ylabel('SSE (Inertia)')
```

```
plt.legend()
```

```
plt.grid(True)
```

```
plt.show()
```



Parameters of k-means

- ① n-clusters → number of clusters
- ② init → method used for initializing clusters.
- ③ n-init → number of times the algorithm will be initialized with different centroids. often initialization see
- ④ max_iter : Maximum number of iterations for each the k-means algorithm. The algorithm stops if it is reached before this number
- ⑤ random_state (or seed) : controls the random number for centroid initialization.

Code

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import random

# Step 1: Create synthetic income dataset
np.random.seed(42)
names = [f'Person_{i}' for i in range(1, 51)]
ages = np.random.randint(20, 60, size=50)
incomes = np.random.randint(20000, 120000, size=50)

income_df = pd.DataFrame({
    'Name': names,
    'Age': ages,
    'Income': incomes
})

# Save to CSV (optional)
income_df.to_csv("income.csv", index=False)

# Step 2: Load and preprocess
df = pd.read_csv("income.csv")

# Drop name (non-numeric)
df_numeric = df.drop('Name', axis=1)

# Step 3: Scaling
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_numeric)

# Step 4: Train-test split
X_train, X_test = train_test_split(df_scaled, test_size=0.2, random_state=42)

# Step 5: SSE vs number of clusters (Elbow method)
```

```

sse = []
k_range = range(1, 11)
for k in k_range:
    km = KMeans(n_clusters=k, random_state=42)
    km.fit(X_train)
    sse.append(km.inertia_)

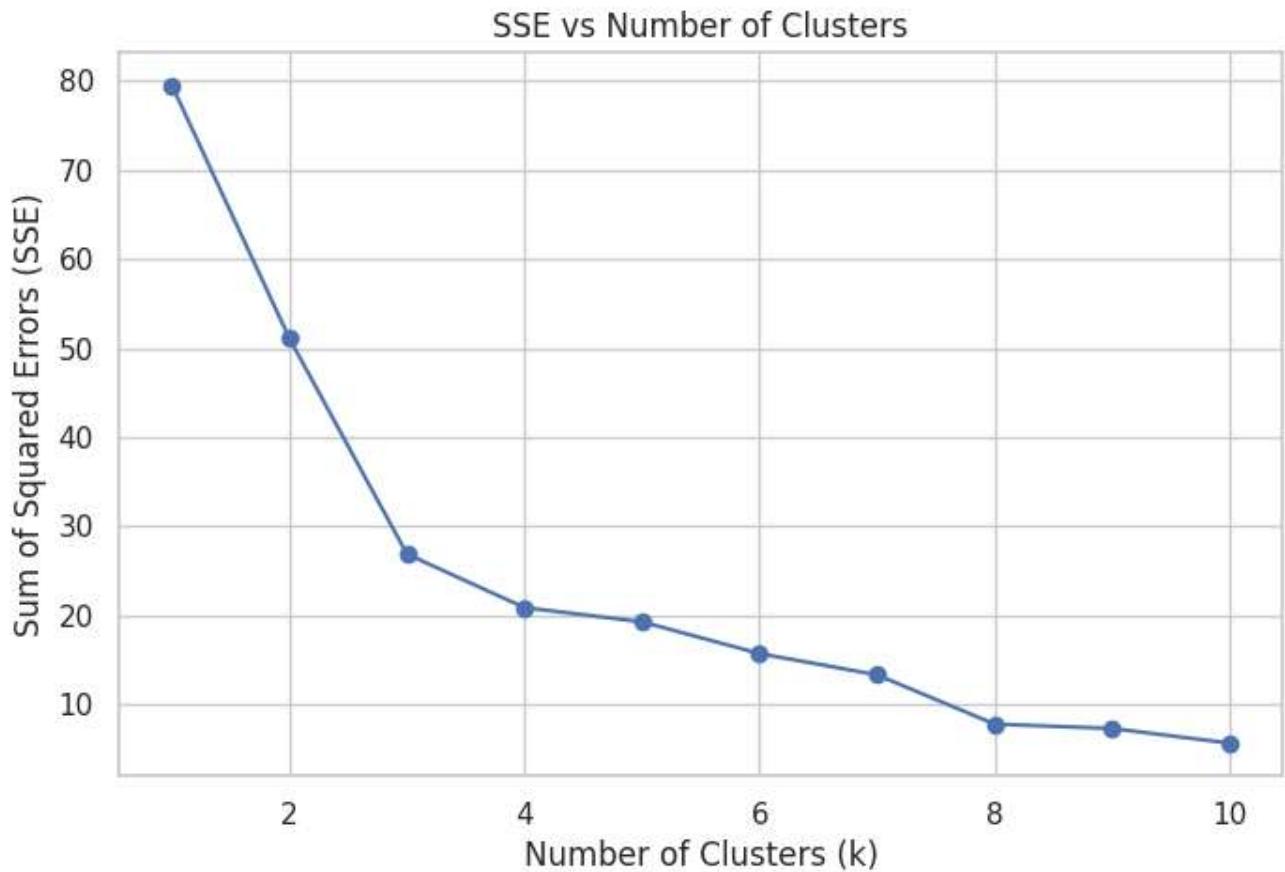
# Step 6: Plot SSE
plt.figure(figsize=(8, 5))
plt.plot(k_range, sse, marker='o')
plt.title('SSE vs Number of Clusters')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Sum of Squared Errors (SSE)')
plt.grid(True)
plt.show()

# Step 7: Fit KMeans with optimal k (e.g., 3)
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans.fit(X_train)

# Step 8: Predict clusters on test set
y_pred = kmeans.predict(X_test)

# Step 9: "Accuracy" – not meaningful in unsupervised, but just a check
print("Cluster labels for test data:", y_pred)

```



Cluster labels for test data: [2 1 1 1 2 0 2 2 1 0]

In [14]:

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris

# Step 1: Load iris dataset and use only petal length & width
iris = load_iris()
iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)
iris_df = iris_df[['petal length (cm)', 'petal width (cm)']] # drop other features

# Step 2: Scaling
scaler = StandardScaler()
iris_scaled = scaler.fit_transform(iris_df)

# Step 3: Elbow method to find optimal k

```

```

sse = []
k_range = range(1, 11)
for k in k_range:
    km = KMeans(n_clusters=k, random_state=42)
    km.fit(iris_scaled)
    sse.append(km.inertia_)

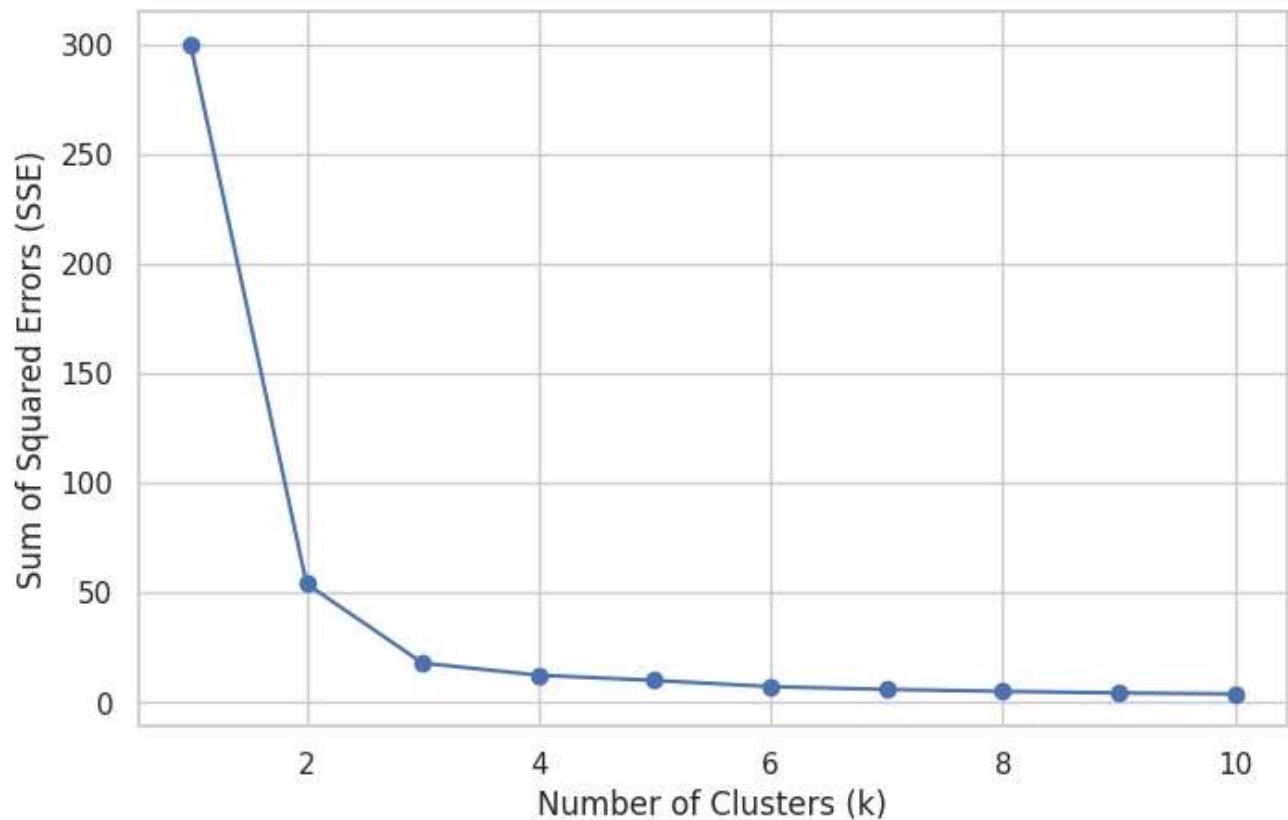
# Step 4: Plot SSE vs k
plt.figure(figsize=(8, 5))
plt.plot(k_range, sse, marker='o')
plt.title("Elbow Method: SSE vs Number of Clusters")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Sum of Squared Errors (SSE)")
plt.grid(True)
plt.show()

# Optional: Fit with optimal k = 3 and visualize clusters
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
labels = kmeans.fit_predict(iris_scaled)

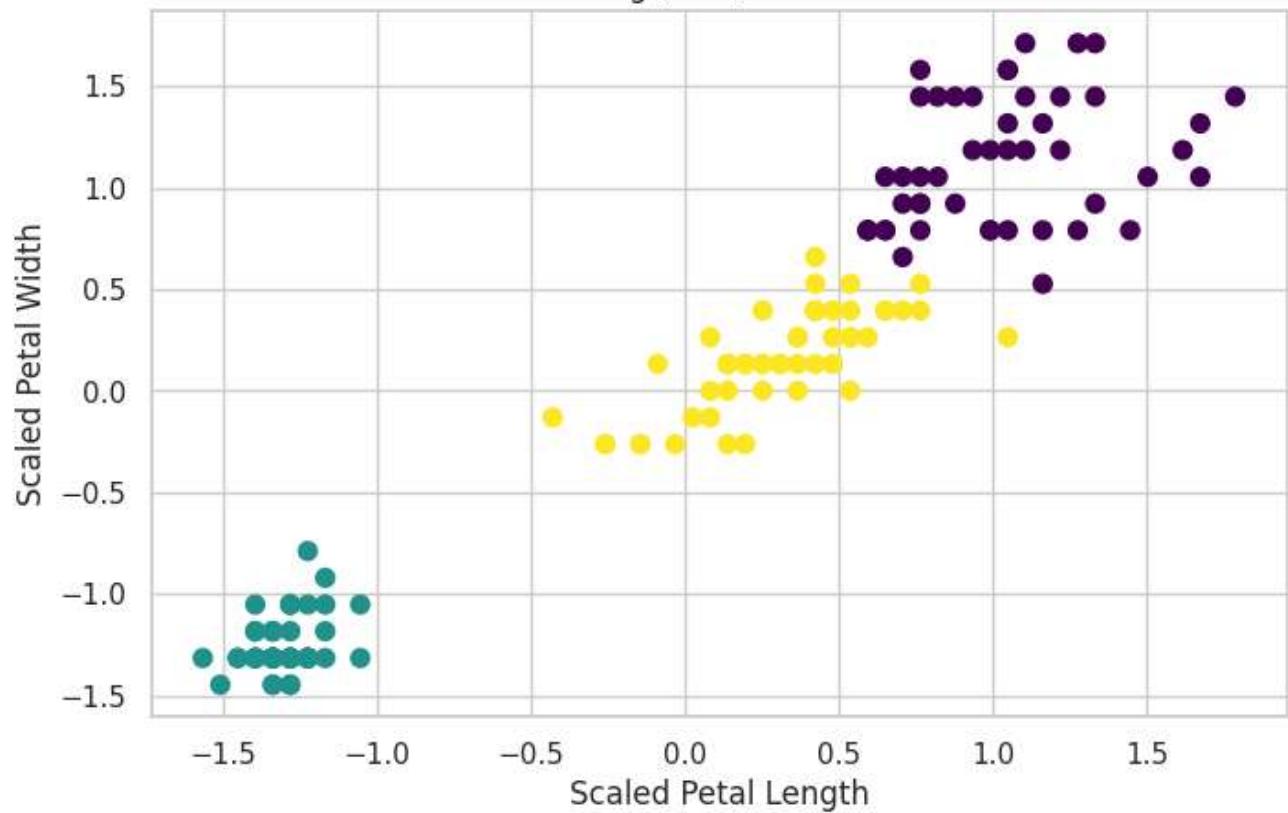
plt.figure(figsize=(8, 5))
plt.scatter(iris_scaled[:, 0], iris_scaled[:, 1], c=labels, cmap='viridis', s=50)
plt.title("KMeans Clustering (k=3) on Iris Petal Features")
plt.xlabel("Scaled Petal Length")
plt.ylabel("Scaled Petal Width")
plt.grid(True)
plt.show()

```

Elbow Method: SSE vs Number of Clusters



KMeans Clustering (k=3) on Iris Petal Features



In []:

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method.

Algorithm

Lab-11 Principal Component Analysis

DATE _____

Given the data in table reduce dimension for using PCA

Feature	Example Number	1	2	3
x_1		4	8	13
x_2		11	4	5

① $\bar{x}_1 = \frac{1}{4} (4+8+13+7) = 8$

$\bar{x}_2 = \frac{1}{4} (11+4+5+14) = 8.5$

$\text{Cov}(x_1, x_1) = \frac{1}{N-1} \sum_{k=1}^N (x_{1k} - \bar{x}_1)^2$

$$= \frac{1}{3} [(4-8)^2 + (8-8)^2 + (13-8)^2 + (7-8)^2]$$

$$= 14$$

$\text{Cov}(x_1, x_2) = \frac{1}{N-1} \sum_{k=1}^N (x_{1k} - \bar{x}_1)(x_{2k} - \bar{x}_2)$

$$= \frac{1}{3} [(4-8)(11-8.5) + (8-8)(4-8.5) + (13-8)(5-8.5) + (7-8)(14-8.5)]$$

$$= -11$$

~~$\text{Cov}(x_2, x_1) = \text{Cov}(x_1, x_2)$~~

$= -11$

$\text{Cov}(x_2, x_2) = \frac{1}{N-1} \sum_{k=1}^N (x_{2k} - \bar{x}_2)^2$

Characteristic equation =

$$0 = \det(s - \lambda I)$$

$$0 = \begin{vmatrix} 14-\lambda & -11 \\ -11 & 23-\lambda \end{vmatrix}$$

$$= (14-\lambda)(23-\lambda) - (-11)(-11)$$

$$0 = \lambda^2 - 37\lambda + 201$$

$$\lambda = \frac{1}{2}(37 \pm \sqrt{565})$$

Eigen vector corresponding to $\lambda = \lambda_1$ is a vector

$$U = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = (s - \lambda_1 I)X$$

$$= \begin{bmatrix} 14-\lambda_1 & -11 \\ -11 & 23-\lambda_1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} (14-\lambda_1)u_1 - 11u_2 \\ -11u_1 + (23-\lambda_1)u_2 \end{bmatrix}$$

using theory of system of linear equations
equations are not independent.

$$\frac{u_1}{11} = \frac{u_2}{14-\lambda_1} = t,$$

$$u_1 = 11t, \quad u_2 = (14-\lambda_1)t.$$

Taking $t = 1$, we get an eigenvector

λ_1 as

code

```
import numpy as np
from sklearn.decomposition import PCA
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LogisticRegression
from sklearn.metrics import accuracy_score

digits = load_digits()
X = digits.data
y = digits.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)

pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.fit_transform(X_test_scaled)

log_reg = LogisticRegression(max_iter=10000)
log_reg.fit(X_train_pca, y_train)

y_pred = log_reg.predict(X_test_pca)

accuracy = accuracy_score(y_test, y_pred)
```

Let

DAT

PCA Algorithms

- 1) Calculate mean
- 2) calculate covariance matrix
- 3) Eigen value of covariance matrix
- 4) Computation of eigen vectors - unit of eigenvectors
- 5) Computation of first principle component
- 6) Geometric meaning of 1st principle component

N
19/5/26

Code

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score

# Load dataset
digits = load_digits()
X, y = digits.data, digits.target

# Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Logistic Regression without PCA
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)
preds = lr.predict(X_test)
print(f"Accuracy without PCA: {accuracy_score(y_test, preds):.4f}")

# Apply PCA with 2 components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Split PCA-reduced data
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y, test_size=0.2, random_state=42)

# Logistic Regression with PCA-reduced data
lr_pca = LogisticRegression(max_iter=1000)
lr_pca.fit(X_train_pca, y_train_pca)
preds_pca = lr_pca.predict(X_test_pca)
```

```
print(f"Accuracy with PCA (2 components): {accuracy_score(y_test_pca, preds_pca):.4f}")
```

Accuracy without PCA: 0.9722

Accuracy with PCA (2 components): 0.5389

In [13]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
from scipy.stats import zscore

# Load dataset
heart_df = pd.read_csv("heart.csv")

# Print column names to verify
print("Column names in dataset:", heart_df.columns.tolist())

# Optional: rename columns if necessary (based on your dataset)
# Example only – skip this if your columns are already clean
heart_df.columns = [col.strip().lower() for col in heart_df.columns]

# Reprint cleaned column names
print("Cleaned column names:", heart_df.columns.tolist())

# Remove outliers using Z-score
z_scores = np.abs(zscore(heart_df.select_dtypes(include=[np.number])))
heart_df = heart_df[(z_scores < 3).all(axis=1)]

# Determine categorical columns that need encoding
categorical_cols = heart_df.select_dtypes(include=['object', 'category']).columns.tolist()

# Encode categorical columns using one-hot encoding
if categorical_cols:
    heart_df = pd.get_dummies(heart_df, columns=categorical_cols, drop_first=True)
```

```

# Split features and target
target_col = 'target' if 'target' in heart_df.columns else heart_df.columns[-1]
X = heart_df.drop(target_col, axis=1)
y = heart_df[target_col]

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Models
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'SVM': SVC(),
    'Random Forest': RandomForestClassifier()
}

print("\nModel Accuracy WITHOUT PCA:")
for name, model in models.items():
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    print(f'{name}: {accuracy_score(y_test, preds):.4f}')

# Apply PCA
pca = PCA(n_components=5)
X_pca = pca.fit_transform(X_scaled)
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y, test_size=0.2,
random_state=42)

print("\nModel Accuracy WITH PCA:")
for name, model in models.items():
    model.fit(X_train_pca, y_train_pca)
    preds = model.predict(X_test_pca)
    print(f'{name}: {accuracy_score(y_test_pca, preds):.4f}')

```

Column names in dataset: ['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS', 'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope', 'HeartDisease']

Cleaned column names: ['age', 'sex', 'chestpaintype', 'restingbp', 'cholesterol', 'fastingbs', 'restingeccg', 'maxhr', 'exerciseangina', 'oldpeak', 'st_slope', 'heartdisease']

Model Accuracy WITHOUT PCA:

Logistic Regression: 0.9444

SVM: 0.9333

Random Forest: 0.9222

Model Accuracy WITH PCA:

Logistic Regression: 0.8611

SVM: 0.8778

Random Forest: 0.8833

In []: